175 PTAS
mi computer
56
CURSO PRACTICO DEL ORDENADOR PERSONAL,
EL MICRO Y EL MINIORDENADOR
Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen V - Fascículo 56

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, 08008 Barcelona
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-007-4 (tomo 5)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 068502
Impreso en España - Printed in Spain - Febrero 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S.A. (Paseo de Gracia, 88, 5.°, 08008 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

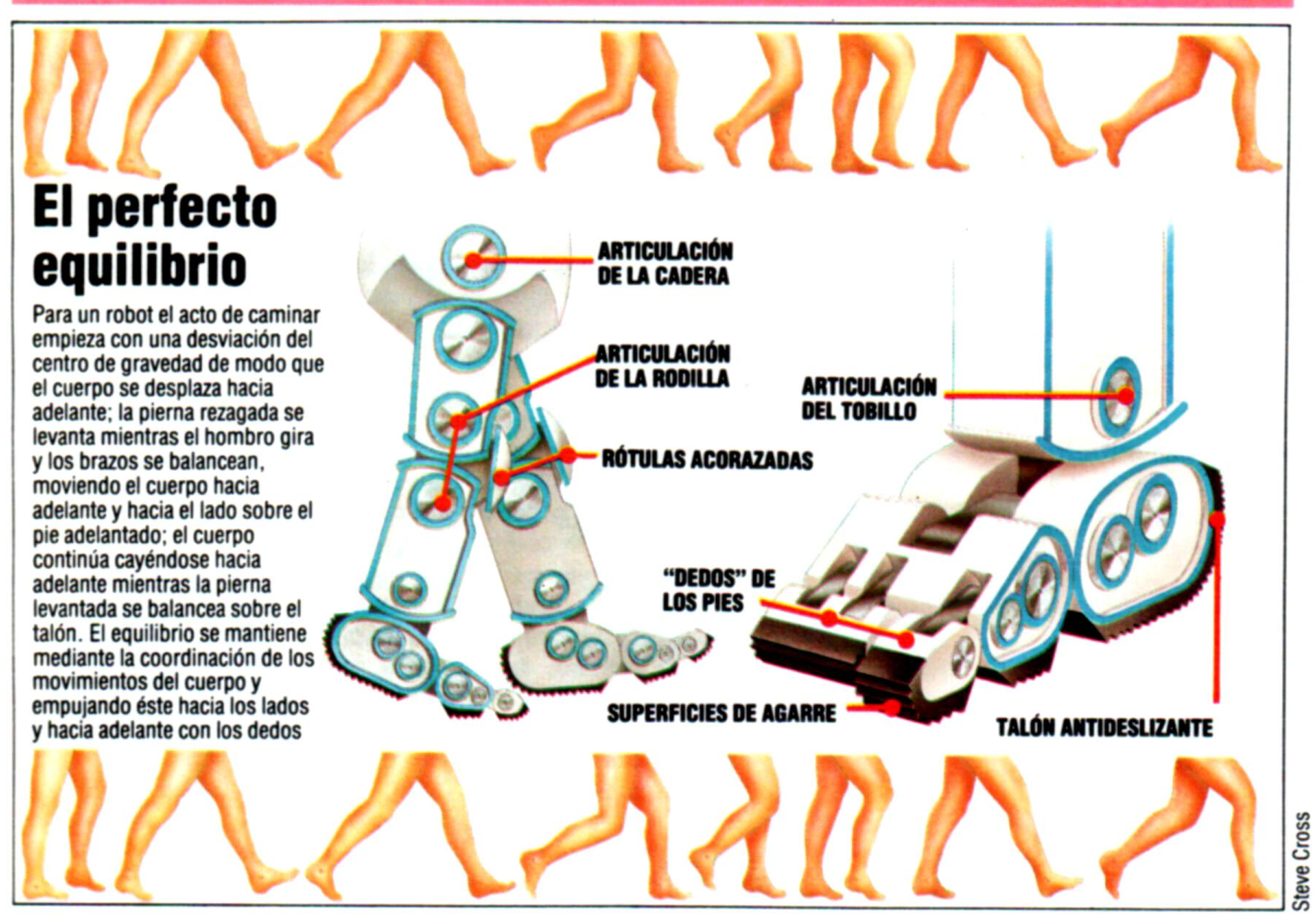
No se efectúan envíos contra reembolso.





¡En marcha!

En esta ocasión analizaremos los tres métodos principales de movimiento del robot y el medio más eficaz de controlarlo



El perfecto equilibrio

Para un robot el acto de caminar empieza con una desviación del centro de gravedad de modo que el cuerpo se desplaza hacia adelante; la pierna rezagada se levanta mientras el hombro gira y los brazos se balancean, moviendo el cuerpo hacia adelante y hacia el lado sobre el pie adelantado; el cuerpo continúa cayéndose hacia adelante mientras la pierna levantada se balancea sobre el talón. El equilibrio se mantiene mediante la coordinación de los movimientos del cuerpo y empujando éste hacia los lados y hacia adelante con los dedos

Steve Cross

Mucho antes de que un niño aprenda a caminar, puede coger objetos y demostrar su inteligencia de numerosas maneras; pero caminar es una habilidad cuyo desarrollo lleva mucho tiempo y exige una considerable práctica antes de que adquiera un carácter automático.

Se puede hacer que los robots "caminen", pero las técnicas implicadas son muy distintas de los métodos que emplean los seres humanos. El robot puede tener piernas, capaces de balancearse hacia adelante y hacia atrás en una aproximación al caminar humano, pero cada una de ellas está provista de un pie que posee ruedas en su base. Estas ruedas están equipadas con mecanismos para impedir el movimiento de retroceso. De modo que un robot de este tipo sigue, cuando "camina", una secuencia de acciones preestablecida. La desventaja de este procedimiento es que es difícil desarrollar una forma de gobernar el robot; éste tenderá a moverse sólo hacia adelante y, además, sus movimientos serán imprecisos.

Una solución mucho mejor sería hacer que los robots caminaran levantando primero una pierna y luego la otra, como hacen los humanos, en vez de balancear simplemente cada extremidad a través de un arco limitado. El principal problema de este enfoque es que el robot debe ser capaz de mantenerse en equilibrio sobre una sola pierna mientras va caminando. Se han probado diversas soluciones: éstas incluyen inclinar el cuerpo del robot hacia los lados sobre un riel de modo que el centro de gravedad del robot se sitúe directamente encima de la pierna que está soportando su peso. Si se desarrollara un sistema de este tipo, los robots podrían caminar de una forma eficaz. En teoría, se podría diseñar un robot que subiera las escaleras y le llevara a su dueño el desayuno por la mañana. Pero en la práctica, a pesar de que un robot que suba las escaleras es bastante factible, ya sería más difícil desarrollar un robot que "supiera" cuándo terminaba la escalera, debido a los aparatos adicionales necesarios para detectar el último escalón.

Un enfoque alternativo ha sido el de montar robots sobre cadenas similares a las de los tanques. Este sistema ofrece la ventaja de permitir que el robot se mueva sobre terreno irregular. Los ejércitos más modernos utilizan este tipo de robots oruga para llevar a cabo peligrosos trabajos de desactivación de bombas; estas máquinas pueden maniobrar a través de escombros y son capaces de cubrir un terreno moderadamente desnivelado.

Las cadenas son sólidas y, a la vez, fáciles de dirigir, pero poseen dos inconvenientes fundamentales. El primero es que, dado que la mayoría de los robots son bastante pequeños, también son reducidas las dimensiones de las cadenas y, en consecuencia, no se pueden salvar grandes obstáculos. Un tanque de combate de tamaño natural puede abrirse paso prácticamente ante cualquier obstáculo con facilidad; pero esto se debe sólo a que los tanques son grandes, pesados y potentes. Si un carro de combate tratara de pasar sobre un objeto tan gran-

de que su centro de gravedad se desplazara fuera de la zona de sus cadenas, se caería. Esto de hecho sucede a veces cuando el terreno es demasiado irregular. Lo mismo le sucedería a un robot oruga si intentara subir por superficies demasiado escarpadas.

La segunda desventaja es que las cadenas no se pueden controlar con precisión. El mecanismo de dirección consiste en detener una cadena de modo que la otra continúe en movimiento; el robot (o el tanque) se desplaza, por consiguiente, describiendo un arco. Cuando esto sucede, la banda estacionaria puede que se desplace ligeramente y la posición final no será la esperada. Un carro de combate conducido por una persona puede corregir rápidamente cualquier error de este tipo; pero para un robot las necesarias correcciones del recorrido son considerablemente más complejas.

Para el control del robot, obviamente es deseable contar con un juego de instrucciones que siempre hagan que el robot se desplace exactamente hasta el lugar correcto, encaminándolo en una dirección predecible con exactitud. Precisamente por este motivo la forma más común de movimiento de los robots es mediante ruedas. Éstas poseen varias ventajas evidentes, al ser simples, eficaces y capaces de producir un movimiento mucho más uniforme que el que jamás se podría obtener con las piernas.

Una vez aceptado que el robot ha de tener ruedas, el único problema es el control exacto del movimiento. Consideremos, por ejemplo, un automóvil de juguete accionado por cuerda. Éste va sobre ruedas pero no es un robot, dado que no posee ningún medio que le permita "conocer" su posición en cualquier momento dado. Lo que se necesita es un sistema de coordenadas que se pueda utilizar para determinar la posición de un objeto sobre una superficie; el sistema más común a este fin se vale de coordenadas cartesianas. Con este sistema es posible determinar la posición exacta de un robot y especificar los movimientos que se precisan para desplazarlo hasta otra posición definida. Todo lo que se necesita luego es un dispositivo para asegurar que el robot se pueda mover con precisión dentro de este marco de referencia.

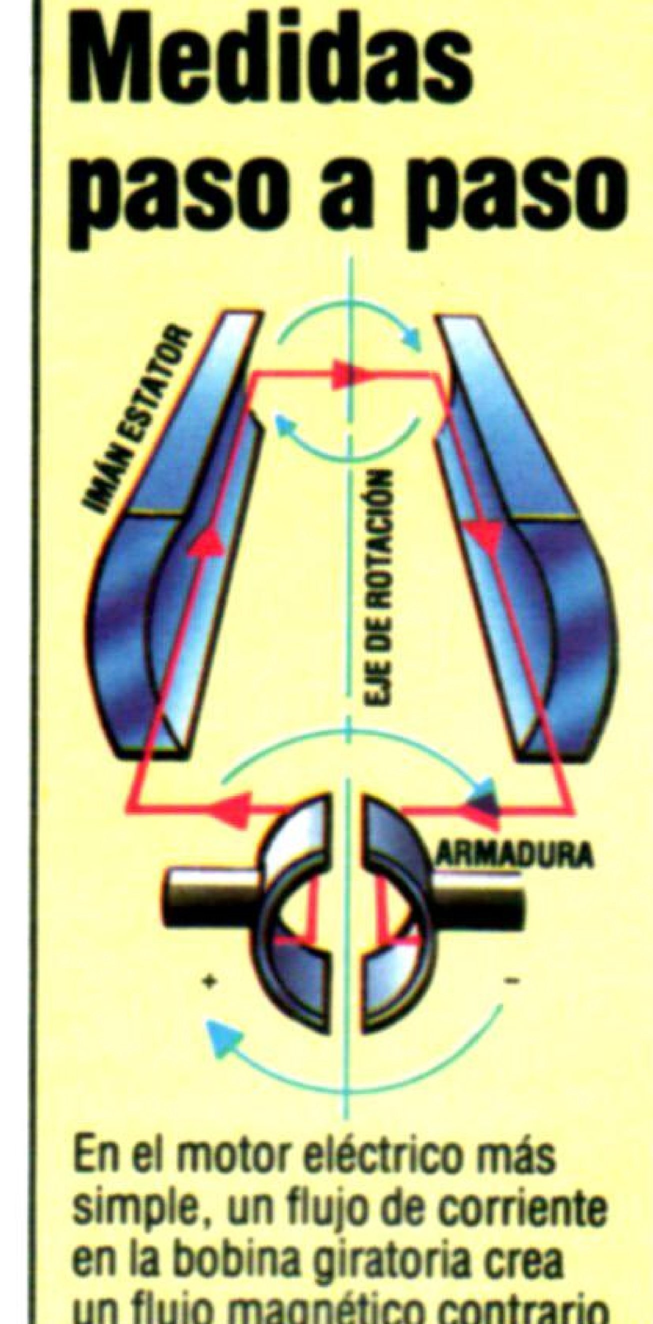
Si bien ocasionalmente se utiliza la energía hidráulica o neumática, el método más común para desplazar robots móviles es mediante un motor eléctrico. Como hemos visto en nuestro apartado de *Bricolaje* (pp. 1065 y 1092), un motor eléctrico sencillo puede proporcionar movimiento y un discreto control sobre la dirección. El mismo no es apto para un control de precisión; un motor eléctrico sencillo siempre efectúa un giro de al menos 180° antes de un detención, y con frecuencia la inercia hace que rote un poco más que eso.

Por consiguiente, para el control de robots normalmente se utiliza un motor paso a paso. Éste contiene una gran cantidad de bobinas y, si bien los diseños pueden variar enormemente, el principio general del motor paso a paso permite especificar porciones de rotación pequeñas y exactas, con poco error por exceso (rotar más de lo que debería) o por defecto (rotar demasiado poco).

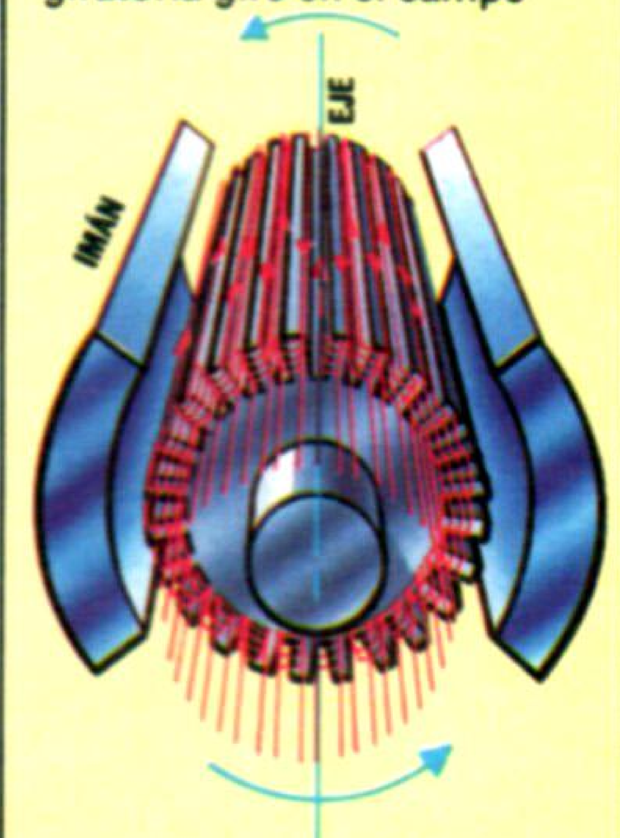
Los robots que emplean motores paso a paso están ampliamente difundidos. Tales robots suelen tener acoplado un lápiz que les permite trazar una línea sobre la superficie sobre la cual se están desplazando. Estos robots "esgrimidores" de lápices se llaman *tortugas* y se dice que los diseños que producen son *gráficos tortuga*. Están capacitados para efectuar un movimiento preciso; su precisión se puede juzgar instruyéndolos para que dibujen una forma cerrada, como un rectángulo o una estrella, comprobando si la línea trazada se cierra o no sobre sí misma en el punto de comienzo.

Los motores paso a paso y las coordenadas cartesianas, por consiguiente, nos pueden proporcionar un método relativamente preciso de controlar el movimiento de un robot. No obstante, si el robot ha de hacer algo más que simplemente dar vueltas por encima de una superficie dada, tropezando con objetos, necesitará ser capaz de responder con rapidez y precisión a las condiciones externas. En el próximo capítulo nos centraremos en ello.

Medidas paso a paso



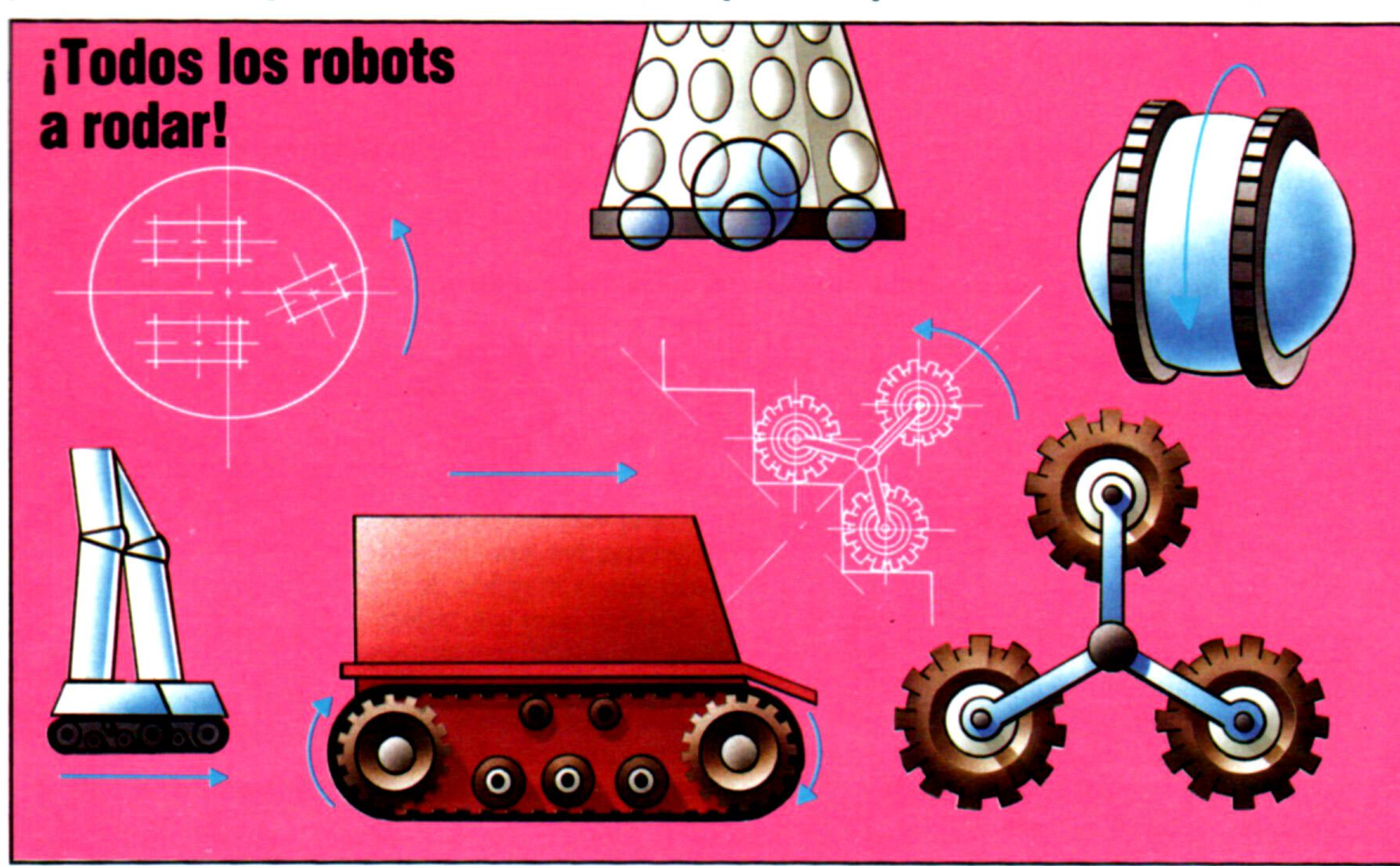
En el motor eléctrico más simple, un flujo de corriente en la bobina giratoria crea un flujo magnético contrario al del campo magnético del imán estator; esta oposición de fuerzas hace que la pieza giratoria gire en el campo



El motor paso a paso posee muchas bobinas, a veces centenares. La conmutación de corriente de una bobina a otra hace que el conjunto gire en incrementos de arco controlables exactamente

Kevin Jones

¡Todos los robots a rodar!



He aquí algunos de los posibles escenarios para el desplazamiento de un robot. Las piernas con cadenas o correas ofrecen agarre sacrificando maniobrabilidad, pero permiten una marcha uniforme sin elevación de piernas, minimizando, por consiguiente, los problemas relativos al equilibrio. Los vehículos robot con este sistema son muy corrientes en los equipos de exploración planetaria y en las patrullas desactivadoras de bombas.
El formato de tres ejes es la única adaptación que permite al robot subir escalones.
Una gran rueda que gire rodeada de estabilizadores es muy fácil de gobernar, pero es sensible a las superficies irregulares. La disposición de dos ruedas fijas y una dirigida es el mínimo necesario para la estabilidad.
Distribuir la carga sobre ruedas motrices es acertado, pero eleva el centro de gravedad y reduce la estabilidad

Kevin Jones



Divide y vencerás

En este capítulo de nuestro curso de LOGO crearemos unos interesantes gráficos basados en la utilización de los procedimientos denominados "recursivos"

Nuestro primer programa está diseñado para dibujar formas de "árbol". Para comenzar, podemos simplemente dibujar un tronco con una rama a la izquierda y una rama a la derecha. Luego éstas se pueden formar exactamente de la misma manera (si bien serán más pequeñas), con una rama principal central y ramitas a la derecha y a la izquierda. Si se continuara este proceso se crearía gradualmente una forma de árbol.

Éste es un buen ejemplo de cómo se puede utilizar la recursión en LOGO.

Nuestro procedimiento para dibujar este árbol "binario" exige dos entradas: una para la longitud del tronco y la otra para el número de "nivel". La longitud de las ramas se reduce a la mitad a cada nivel, alejándose del tronco.

```
TO RAMA :LENGTH :NIVEL
   IF :NIVEL=0 THEN STOP
   FD :LENGTH
   LT 45
   RAMA (:LENGTH/2)(:NIVEL-1)
   RT 90
   RAMA (:LENGTH/2)(:NIVEL-1)
   LT 45
   BK :LENGTH
END
```

Observe que el procedimiento es transparente. Esto es importante, puesto que, de lo contrario, el "estado" de la tortuga (su posición y encabezamiento) cambiaría cada vez que el procedimiento se llamara a sí mismo, haciendo imposible continuar con el dibujo.

Hay que admitir que este procedimiento no produce un árbol realista; para hacerlo más interesante, se puede modificar el procedimiento de diversas maneras.

Ahora proponemos una versión que dibuja tres ramas, cada una de ellas de diferente longitud, en cada nivel:

```
TO RAMA1:LENGTH :NIVEL
   IF :NIVEL=0 THEN STOP
   FD :LENGTH
   LT 30
   RAMA1(:LENGTH/3)(:NIVEL-1)
   RT 40
   RAMA1(:LENGTH/2)(:NIVEL-1)
   RT 50
   RAMA1 (:LENGTH/1.5)(:NIVEL-1)
   LT 60
   BK :LENGTH
END
```

Pruebe otras modificaciones para producir árboles más parecidos a la realidad.



Polígonos a cuadros

El siguiente procedimiento dibuja un cuadrado, lo divide en cuatro, luego divide cada parte en cuatro, y así sucesivamente:

```
TO TABLERO :LENGTH :NIVEL
   IF :NIVEL=0 THEN REPEAT 4 [FD :LENGTH RT 90]
      STOP
   TABLERO (:LENGTH/2)(:NIVEL-1)
   FD (:LENGTH/2)
   TABLERO (:LENGTH/2)(:NIVEL-1)
   RT 90
   FD (:LENGTH/2)
   LT 90
   TABLERO (:LENGTH/2)(:NIVEL-1)
   BK (:LENGTH/2)
   TABLERO (:LENGTH/2)(:NIVEL-1)
   LT 90
   FD (:LENGTH/2)
   RT 90
END
```

Escriba un procedimiento similar que divida un triángulo en cuatro más pequeños, que luego divida éstos en otros cuatro, y así sucesivamente.

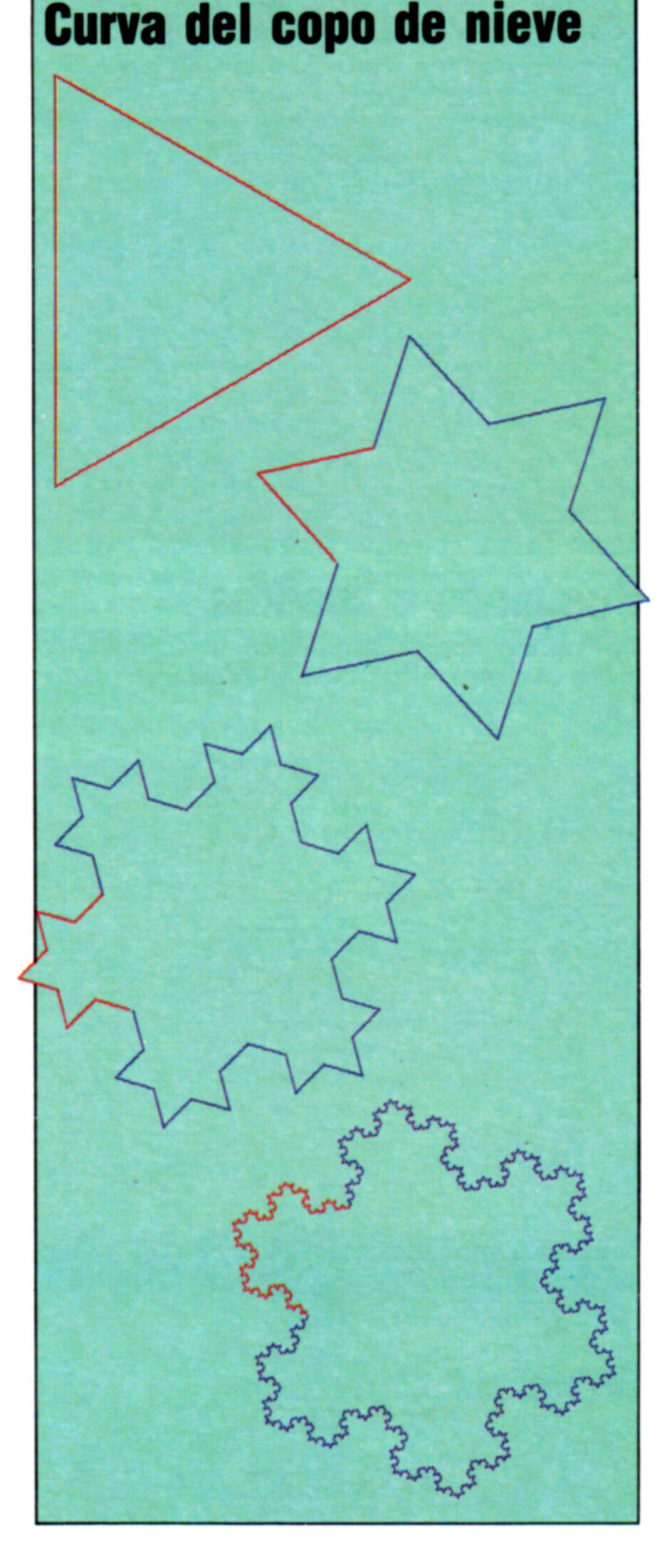
Copos de nieve

Dibuje primero un triángulo equilátero. Divida cada lado en tres partes iguales y dibuje un nuevo triángulo equilátero en la sección central. Borre las líneas compartidas, luego repita esta secuencia para

cada lado de la nueva forma, y continúe el proceso. Se dice que la forma resultante es la "curva del copo de nieve", debido a su aspecto.

```
TO NIEVE :TAMAÑO :NIVEL
   REPEAT 3 [LADO :TAMAÑO :NIVEL RT 120]
END

TO LADO :TAMAÑO :NIVEL
   IF :NIVEL=0 THEN FD :TAMAÑO STOP
   LADO (:TAMAÑO/3)(:NIVEL-1)
   LT 60
   LADO (:TAMAÑO/3)(:NIVEL-1)
   RT 120
   LADO (:TAMAÑO/3):(NIVEL-1)
   LT 60
   LADO(:TAMAÑO/3)(:NIVEL-1)
END
```



Observe que LADO no es transparente, sino que, en cambio, se ha construido como para dejar a la tortuga en el lugar adecuado para dibujar el lado siguiente.

Si este proceso de división se continuara indefinidamente (los matemáticos utilizan la frase "hasta el límite"), el resultado sería una curva que posee una longitud infinita y que, sin embargo, ¡delimita una superficie finita (fija)! Se puede demostrar que esta curva no es ni unidimensional ni bidimensional, sino que está más bien en un punto intermedio entre ambos extremos.

Se podría construir una curva similar comenzando con un cuadrado, dividiendo cada lado en tres partes iguales, construyendo cuadrados en las secciones del medio, y así sucesivamente. Trate de escribir un procedimiento que haga esto.

La serie de curvas que podemos observar en la página contigua la inventó un matemático llamado Sierpinski y sirven para rellenar espacios. Si el proceso se continúa hasta el límite, el resultado es una curva (una línea unidimensional) que pasa a través de cada uno de los puntos del cuadrado que la rodea (una forma bidimensional). Existen muchas otras "curvas para rellenar espacios" que tienen este curioso comportamiento.

El procedimiento empleado para dibujar esta curva es bastante complicado. La curva del nivel 1 se compone de cuatro lados (coloreados en azul) que se unen mediante cuatro diagonales (coloreadas de rojo). De modo que el procedimiento principal, SIERP, no hace más que dividir el proceso en cuatro secciones para que el procedimiento UN.LADO las trate de una en una.

Consideremos sólo uno de los lados. Éste se compone de tres líneas: una diagonal, otra horizontal o vertical y una tercera diagonal. En el nivel 2, cada diagonal se sustituye por otro conjunto, más pequeño, de tres líneas, y la línea horizontal o vertical se reemplaza por dos conjuntos similares de tres líneas unidos mediante una línea. Para pasar de un nivel a otro se lleva a cabo el mismo proceso.

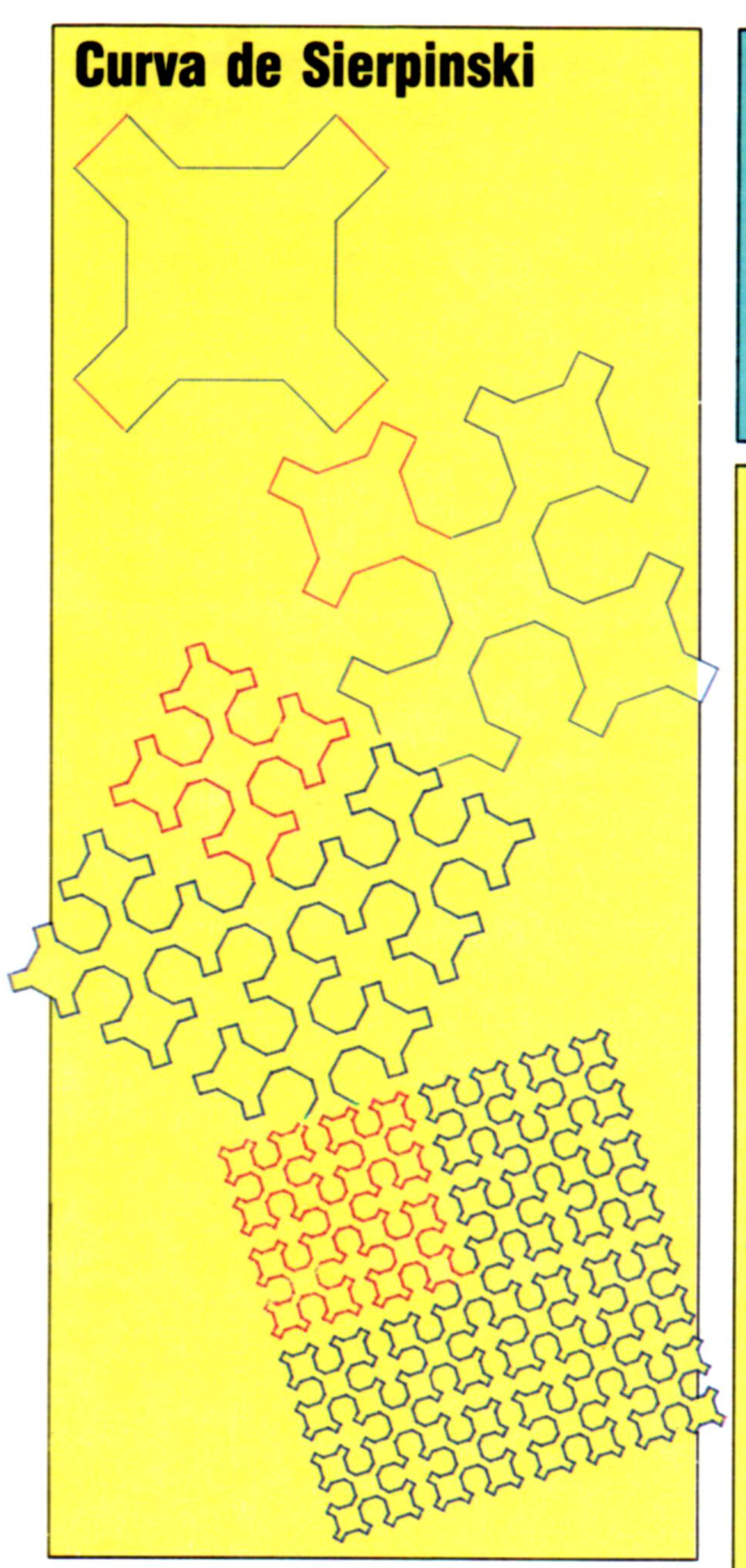
He aquí los procedimientos para dibujar las curvas. Observe cómo se utiliza la instrucción del LOGO MAKE para inicializar DIAG:

```
TO SIERP :LADO :NIVEL
   MAKE "DIAG :LADO / SQRT (2)
   REPEAT 4 [UN.LADO :NIVEL RT 45 FD :DIAG RT 45]
END

TO UN.LADO :NIVEL
   IF :NIVEL=0 STOP
UN.LADO (:NIVEL-1)
   RT 45
   FD :DIAG
   RT 45
   UN.LADO(:NIVEL-1)
   LT 90
   FD :LADO
   LT 90
   FD :LADO
   LT 90
   UN.LADO(:NIVEL-1)
   RT 45
   FD :DIAG
   RT 45
   UN.LADO(:NIVEL-1)
END
```



Curva de Sierpinski





Complementos al LOGO

Las versiones LCSI de LOGO utilizan SETPOS para establecer la posición de la tortuga. Ésta requiere una lista como entrada, de modo que las dos coordenadas se deben combinar con LIST. P. ej.:

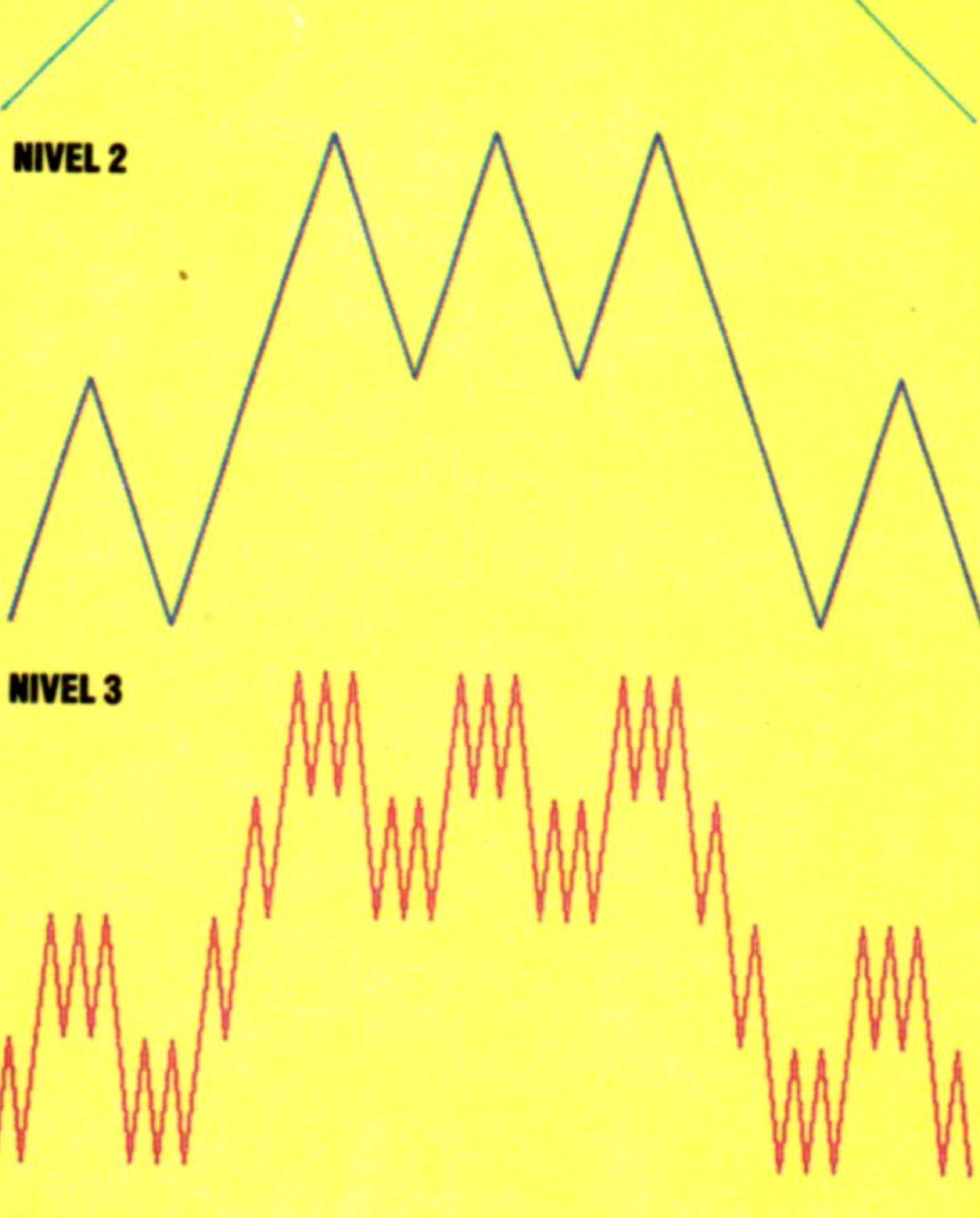
```
SETPOS LIST 45 67
```

Recuerde, asimismo, que las versiones LCSI emplean una sintaxis diferente con IF. Una regla de detención típica sería:

```
IF NIVEL=0 [STOP]
```

Procedimientos (4)

Nuestra ilustración muestra una serie de formas que en su límite definen una curva que no tiene gradiente en ningún punto. El primer nivel se compone de dos líneas: una que va hacia arriba, la otra hacia abajo. Para pasar al siguiente nivel, sustituimos la línea ascendente por una línea quebrada con seis partes. Ésta se eleva hasta la mitad de la altura de la línea original, luego desciende; vuelve a elevarse hasta la mitad, sigue hasta la altura total, cae hasta la media altura y sube hasta la altura total. La descendente se divide en seis secciones.



Intente escribir un conjunto de procedimientos que dibuje esta serie de curvas. Deberá utilizar SETXY en lugar de FD y RT. Su procedimiento de nivel superior habrá de dividir la tarea en dos partes: una para ascender, la otra para descender. Luego tendrá que escribir dos procedimientos separados para tratar con las dos partes del procedimiento. Estos procedimientos se pueden llamar entre sí y a sí mismos.

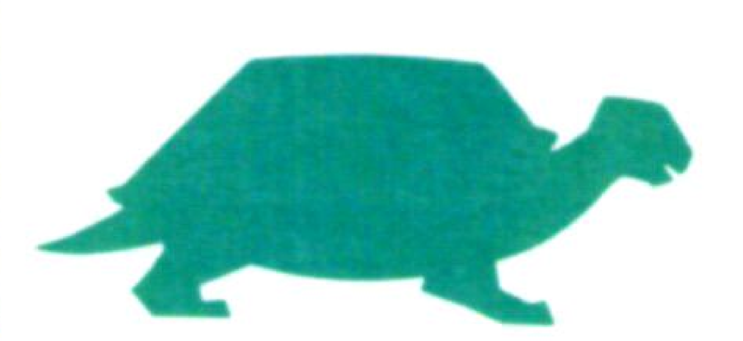
Respuestas a los ejercicios

Procedimiento que dibuja una torre de cuadrados:

```
TO TORRE :TAMAÑO
   IF:TAMAÑO<5 THEN STOP
   CUADRADO :TAMAÑO
   MOVER :TAMAÑO
   TORRE (:TAMAÑO/2)
END

TO CUADRADO :TAMAÑO
   REPEAT 4 [FD :TAMAÑO RT 90]
END

TO MOVER :TAMAÑO
   FD :TAMAÑO
   RT 90
   FD (:TAMAÑO/4)
   LT 90
END
```



La suma de las partes

Iniciamos una serie en la que analizaremos paquetes de software integrado. Veamos, en primer lugar, qué significa este término

La integración representa una de las tendencias más interesantes que ha habido en el software. Por el momento se aplica principalmente a los sistemas de gestión, aunque ya sus técnicas se han empezado a introducir en los micros personales. Un ejemplo de ello es el Sinclair QL, cuyos cuatro paquetes de software incorporan los principios básicos de la integración (véanse pp. 982-983).

El logro fundamental de la integración es el de permitir que el programador pase de un paquete a otro diferente de manera rápida y fácil. En un sistema ideal no se tendría que salir de un programa, regresar al sistema operativo, intercambiar discos y después comenzar con otro programa. Para que sea eficaz, el cambio de aplicación ha de concretarse casi en pulsar sólo una tecla, y algunos programas, como el *Lotus 1-2-3* y el *Framework*, de Ashton Tate, lo consiguen.

También resulta útil poder transferir datos fácilmente entre paquetes. Por ejemplo, se podría crear una columna de cifras de ventas anuales para su negocio con el programa de hoja electrónica, luego transferir entera esa columna al programa para tratamiento de textos, con el que escribiría el informe anual. Se podrían utilizar los nombres y las direcciones de un archivo de base de datos con el procesador de textos para escribirles una carta personalizada a todas las personas del archivo. En el Lisa y el Macintosh esta facilidad se amplía hasta el punto de que uno puede crear un dibujo a mano alzada con el programa para gráficos y después llevarlo directamente a un documento elaborado con tratamiento de textos.

Además, todos los distintos programas deberían trabajar de la misma forma y producir similar sensación al ser utilizados. Los trazados de pantalla, las teclas de instrucciones, los avisos, los mensajes de error —todos los aspectos de la "interface para el usuario"— deberían ser idénticos o comparables. De no ser así, el usuario no puede pasar con entera confianza de una zona a otra sin tener que detenerse y adaptarse a los cambios en los procedimientos de operatoria. Ello supone una interrupción en la fluidez con que se puede utilizar el software e impide que se lo pueda explotar al máximo.

Un efecto secundario inmediato es que el paquete se vuelve más fácil de aprender. Tener que aprenderse cinco nuevos programas de aplicaciones (algunos activados por menú, otros activados por comandos, todos con distintos formatos de comandos) es una tarea que puede resultarle intimidante a cualquiera. Pero si todos funcionan de la misma manera, el usuario sólo necesita aprenderse uno. Esta característica se conoce como *uniformidad* y se suele citar a menudo cuando se habla de software integrado.

Hemos concluido, entonces, que el software integrado implica tres principios de diseño: facilidad para pasar de una aplicación a otra, libertad para intercambiar datos entre aplicaciones y uniformidad de formato. Estas características contribuyen a hacer que el ordenador sea más accesible al usuario medio, cuyas necesidades se pueden satisfacer con dos o tres aplicaciones de software. Sin duda, contribuirá asimismo a aumentar la popularidad del ordenador personal, dado que su utilización resultará más eficaz y sencilla.

No obstante, el software integrado tiene algunos inconvenientes. El más importante de ellos es el hecho de que los paquetes de software integrado necesitan grandes cantidades de RAM para operar. Imagine el lector tratar de colocar un procesador de textos, una hoja electrónica y una base de datos (las tres aplicaciones que se integran con mayor asiduidad) en 16 o 32 Kbytes. Probablemente se pueda hacer, pero no quedará mucho espacio (en caso de que quede) para almacenar datos. Es esta circunstancia la que limita el software integrado a máquinas con una gran memoria: en general, a ordenadores con 128 Kbytes o más. Por supuesto, los programas que están integrados pueden compartir algunas rutinas, de modo que sólo es necesario escribir una vez las operaciones de almacenamiento en disco y otras tareas del sistema. Sin embargo, cada aplicación tiene sus propias exigencias especiales, y éstas ocupan espacio en RAM.

Una segunda desventaja del software integrado es consecuencia del mismo problema del almacenamiento. Para hacer un ahorro en la cantidad de memoria que requiere un programa, los productores de software recurren a las aplicaciones individuales. Un paquete de proceso de textos que esté incorpo-



Cortesía de Microscope

Ian McKinnell

Para revisar...
Entre los paquetes integrados que estudiaremos en esta serie se incluirán el *Lotus 1-2-3, Open access, Symphony* y *Framework*

rado en un paquete integrado con otras dos o tres aplicaciones no puede ser tan eficaz y completo como un paquete exclusivo. El principal motivo reside en que un programa unitario puede ocupar tanta memoria él solo como todos los programas integrados juntos.

Podemos encontrar un ejemplo en dos programas que se ejecutan en el IBM PC y ordenadores similares. *Multimate* es un programa diseñado alrededor del software que se utiliza en el equipo de proceso de textos exclusivo Wang. Posee muchas opciones para crear y formatear texto de las que no disponen otros programas más pequeños, lo que simplifica de manera considerable la creación de documentos muy extensos y complicados. El *Multimate* requiere, él solo, 192 Kbytes de RAM para operar. El *Lotus 1-2-3*, un programa integrado de tratamiento de textos, hoja electrónica y base de datos, también exige 192 Kbytes. Pero el mismo espacio que utiliza *Multimate* para una aplicación debe ahora contener todo el software necesario para tres programas completos en el *Lotus*. Como resultado, en este último el tratamiento de textos se limita a una simple escritura de informes.

Una tercera desventaja de la integración también se deriva de una de sus ventajas. Es importante, como ya hemos visto, que los programas integrados sean parecidos, de modo que sean fáciles de aprender y de manejar. Lamentablemente, los escritores de software tienen que hacer ciertas concesiones para que ello sea posible. Puede ser que la mejor forma de hacer funcionar una hoja electrónica no sea la más adecuada para hacer operar una base de datos o un procesador de textos, de modo que se tiende a combinar los elementos óptimos de diseño para cada uno en una mezcla utilizable. Microsoft se encontró con este problema cuando diseñó un procesador de textos individual, el Microsoft Word. La empresa deseaba que la visualización en pantalla y la operación del programa fueran compatibles con su hoja electrónica, el Multiplan, que tan enorme éxito había obtenido, para que fuera sencillo integrar los dos programas. Microsoft incluyó en el Word el mismo menú en pantalla que a los usuarios del Multiplan les resultaba tan útil, sólo para descubrir que a los escritores que necesitaban un programa como el Word no les gustaba tener un menú en la pantalla todo el tiempo.

Lo que hay que recordar con respecto al software, es que éste tiene que hacer lo que el usuario desea. Si una persona debe realizar varias tareas, como escribir cartas, llevar cuentas sencillas y tener listas de correspondencia, el software integrado puede hacer que el trabajo sea mucho más fácil. Pero hay que hacer sacrificios, y alguien que desee escribir una novela o informes de empresa muy extensos en su microordenador, quizá tenga que seguir utilizando programas individuales, separados, de tratamiento de textos, hojas electrónicas y bases de datos. No obstante, a medida que los escritores de software vayan aprendiendo más sobre la compresión de instrucciones de ordenador en espacios de memoria cada vez más pequeños, y a medida que la memoria de algunas máquinas personales comience a crecer, el software integrado irá adquiriendo una importancia cada vez mayor tanto para el usuario personal como para el usuario de gestión. A modo de indicio de lo que será el futuro, ya hay dos ordenadores que se están vendiendo con software integrado: el QL de Sinclair y el Plus/4 de Commodore.

En futuros capítulos de esta serie analizaremos algunos de los programas integrados que están teniendo un mayor efecto en el desarrollo de software. Examinaremos dos enfoques distintos a la integración: el que ejemplifican el *Lotus 1-2-3* y el que representan el Lisa y el Macintosh.

Las reglas del juego



Configuración completa
La integración es el principio operativo del Macintosh de Apple, en el cual se produjeron estas ilustraciones. Multiplan (la hoja electrónica), MacWrite (el procesador de textos) y MacPaint (el programa para gráficos) se comunican directamente a través del sistema operativo, de modo que las tres aplicaciones se convierten en una

Código común
Estas visualizaciones de hoja electrónica y tratamiento de textos muestran claramente la uniformidad de formato entre aplicaciones integradas

Cargos transferidos
Los datos se han transferido desde la hoja electrónica al procesador, demostrando la importancia de la compatibilidad y transferencia en el software integrado

Ian McKinnell

Búsqueda de datos

En esta oportunidad consideraremos un procedimiento para buscar un dato específico en una lista de valores

Si en capítulos anteriores hemos mostrado diferentes maneras (burbuja, máximo) de clasificar datos, ya fueran numéricos o alfanuméricos, ahora vamos a mostrar cómo buscar un dato determinado en una lista de valores. Así, contamos con la llamada *búsqueda secuencial*, basada en la comparación de un dato determinado con los contenidos en una tabla, uno por uno. Este proceso puede terminar con dos posibles finales lógicos, a saber: que el dato buscado no se halla en la tabla, y que el dato buscado se encuentra en la tabla y la posición que ocupa.

Presentamos dos modos diferentes de realizar dicha búsqueda; la figura 1 lo hace mediante el uso de contadores en forma tradicional, mientras que la figura 2 lo hace empleando FOR...NEXT.

```
10 REM METODO DE BUSQUEDA SECUENCIAL
20 INPUT "NUMERO DE ELEMENTOS"; N
30 DIM T(N)
40 FOR M=1 TO N
50 INPUT T(M)
60 NEXT M
70 INPUT "VALOR A BUSCAR"; R
80 I=1
90 IF I>N THEN PRINT "VALOR NO
   ENCONTRADO" : END
100 IF R=T(I) THEN PRINT "EL VALOR OCUPA LA
    POSICION";I : END
110 I=I+1
120 GOTO 90
```

Figura 1

INICIO
NÚM. DE ELEMENTOS — N
DIMENSIONAR TABLA T
DESDE M=1 HASTA N
VALOR — Carga de la tabla
SIGUIENTE
VALOR A BUSCAR — R
CONTADOR IGUAL A 1 — I
I>N ? — SÍ: VALOR NO HALLADO → FIN
NO
R=T(I) ? — SÍ: EL VALOR ESTÁ EN LA POSICIÓN I → FIN
NO
SUMAR 1 A CONTADOR I

Figura 2

INICIO
NÚM. DE ELEMENTOS — N
DIMENSIONAR TABLA T
DESDE X=1 HASTA N
VALOR — Carga de la tabla
SIGUIENTE
VALOR A BUSCAR — R
DESDE I=1 HASTA N
R=T(I) ? — SÍ: EL VALOR ESTÁ EN LA POSICIÓN I
NO
SIGUIENTE
VALOR NO HALLADO
FIN



Juguete para artistas

Koala-pad proporciona una forma sencilla de producir gráficos de alta calidad en el Commodore 64

A pesar de su capacidad para producir excelentes efectos gráficos, hasta el momento no se ha dotado al Commodore 64 de los periféricos para gráficos de alta calidad que se han producido para el BBC Micro, por ejemplo. Esto probablemente es consecuencia de las dificultades asociadas con la producción de gráficos en alta resolución en el 64, problemas que han disuadido a los fabricantes de producir tal dispositivo. Ahora, sin embargo, Audiogenic ha empezado a importar un periférico para gráficos fabricado por una empresa norteamericana, Koala Technology, que permite a los usuarios del Commodore 64 un acceso más sencillo a las capacidades para gráficos en alta resolución de la máquina.

A diferencia de tablillas al tacto similares, como la Grafpad (véase p. 649), la Koala-pad es ligera y compacta, dado que sólo mide 20,5×16 cm. En el centro hay un cuadrado de fibra de carbón de 11×11 cm que cubre una membrana sensible al tacto similar al teclado del Spectrum. Presionando simplemente un dedo o un lápiz sobre la membrana, el usuario puede guiar un cursor a través de la pantalla. Esta característica contrasta con otras tablillas para gráficos que exigen un "punzón" especial para completar el circuito.

La membrana está compuesta por dos láminas de cables conductores, una sobre el eje horizontal y otra sobre el eje vertical. Cuando se presiona contra la membrana, el aparato detecta qué cables están en contacto y le envía las coordenadas resultantes al ordenador. Encima de la membrana al tacto hay dos botones, uno de los cuales se debe pulsar cuando el usuario desea colorear un punto de la pantalla o seleccionar una de las diversas opciones para pintar que hay disponibles. Se puede utilizar cualquiera de los dos botones.

La Koala-pad se conecta con el ordenador a través de la puerta para palanca de mando, y el Koala Painter (el software necesario para operar con el dispositivo) se carga desde disco. Una vez cargado, aparece en la pantalla una visualización de las diversas opciones disponibles. En la parte inferior aparece la "paleta", que contiene 16 colores "puros" y 16 colores de "media tinta". Las medias tintas se obtienen coloreando pixels alternos con distintos colores, lo que proporciona el efecto de matizado. Encima de la paleta hay ocho cajas que contienen los "pinceles". Éstos simplemente se componen de diversas formas que se pueden trazar sobre la pantalla, y van desde un único pixel a combinaciones de pixels y líneas. Rodeando los pinceles se hallan las diversas opciones para dibujar líneas o formas en la pantalla. Éstas las selecciona el usuario presionando contra la membrana y dirigiendo de ese modo una flecha de cursor. Cuando la flecha señala la opción deseada, la pulsación de uno de los botones Select la hace entrar en acción. La opción se vuelva intermitente para recordar al usuario cuál es la modalidad que se está utilizando. La Koala-pad proporciona las facilidades para producir líneas individuales, rayos (líneas dibujadas desde un único punto), recuadros y círculos. Se pueden agregar bloques de color mediante la utilización de la opción "caja" (cuadrados coloreados) o "disco" (círculos coloreados). Se obtiene otro tipo de coloreado mediante el empleo de la instrucción FILL para rellenar un área cerrada con el color elegido. Los colores se alteran empleando la instrucción X-COLOUR.

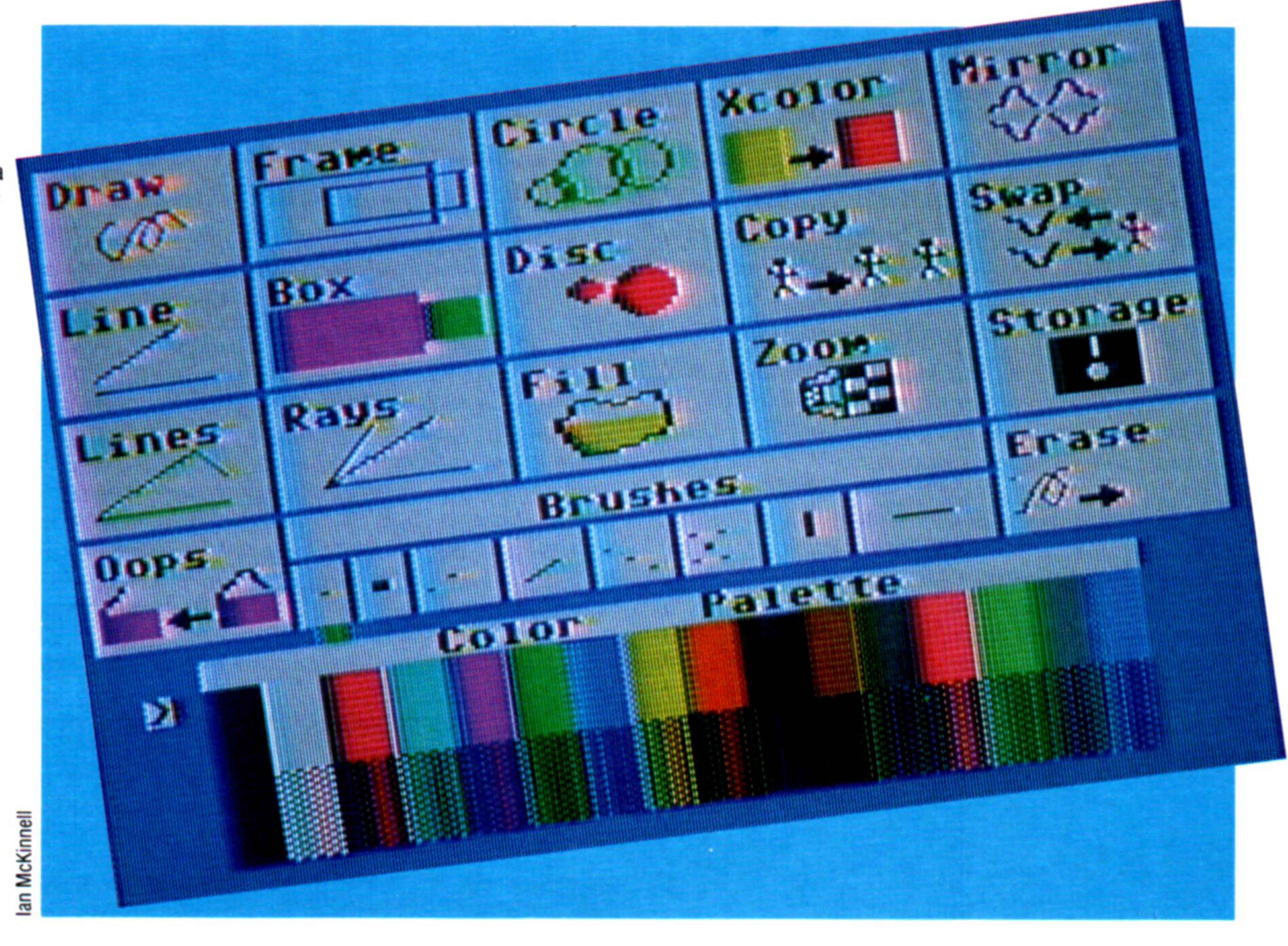
Se pueden dibujar simultáneamente figuras idénticas mediante la utilización del comando MIRROR. Éste divide la pantalla en cuatro secciones, con el cursor restringido al cuarto superior izquierdo. Todo lo que se trace dentro de ese cuadrante se

Para tener en la mano
Empleando el fácilmente comprensible software conducido por menú, en unas pocas horas se pueden construir pantallas de gráficos complejas y sofisticadas. A diferencia de muchas tablillas para gráficos similares, la Koala-pad se puede sostener en la mano mientras se la utiliza



Ian McKinnell

Haga su elección
El menú principal de la Koala-pad se compone de recuadros que contienen tanto el nombre de la función como un icono explicativo. A pesar de que con los iconos se pretende facilitar la comprensión, tal vez algunas de las ilustraciones sean confusas. Se desplaza la flecha del cursor hasta un recuadro y se pulsa el botón "Select". El nombre del recuadro seleccionado se pondrá intermitente para recordar al usuario cuál es la modalidad que está utilizando



Ian McKinnell

copiará automáticamente en el área correspondiente de los otros tres cuadrantes.

Mediante el empleo de la instrucción ZOOM se consigue un trazado de pixels sumamente detallado. El usuario puede elegir cualquier parte de la pantalla, y la misma se visualiza entonces ampliada en una "ventana" en la parte inferior de la pantalla. Los pixels individuales se visualizan como cuadrados del tamaño de un carácter, de ocho por ocho pixels. Gracias a esta característica la producción de tipos y figuras del tamaño de sprites es rápida y sencilla. Estas figuras se colocan luego en cualquier lugar de la pantalla mediante la instrucción COPY; ésta permite que el usuario defina una zona de la pantalla, cuyo contenido se copia en cualquier otra posición.

Después de haber seleccionado estas opciones, la flecha del cursor se desplaza fuera de la pantalla y se pulsa el botón Select. La pantalla presenta entonces el "lienzo" sobre el cual se creará la imagen requerida. Moviendo la flecha del cursor y pulsando el botón Select, se puede dibujar líneas y formas en cualquier lugar de la pantalla normal para gráficos del Commodore 64.

La calidad de los gráficos producidos por este dispositivo es excelente, rivalizando con pantallas de alta resolución producidas con software comercial. Sin embargo, el nivel de calidad de la instrucción DRAW, para dibujar a mano alzada, es decepcionante (calificativo que también comparten otros paquetes para gráficos). La resolución de la matriz de la membrana no se corresponde con la pantalla de alta resolución del 64, de modo que el punzón (o el dedo) del usuario a menudo no se dirigirá directamente sobre una intersección de la cuadrícula y estará, de hecho, activando dos puntos al mismo tiempo. El ordenador, el tratar de interpretar esto, trazará un punto que lamentablemente no siempre estará en la posición que se pretendía. Esto puede llevar a que lo que planificó como una línea recta aparezca como un garabato confuso. Otra crítica es que, aparte de la instrucción ZOOM, no hay ninguna opción para cambiar el color sin tener que retornar al menú principal. Pero éstas son desventajas poco relevantes que la mayoría de las veces pasan a un plano secundario debido a la velocidad con que se ejecutan las instrucciones LINE y FILL.

Otra limitación del software que se podría haber resuelto mejor es el método empleado para borrar los errores. Cuando se produce una equivocación, el error se elimina de la pantalla utilizando la instrucción OOPS, a la cual se accede desde el menú principal. Sin embargo, el empleo de OOPS borrará todo el trabajo que se haya realizado desde que el usuario saliera por última vez del menú principal. Ello significa que tal vez se borre un trabajo de media hora, simplemente debido a un solo error. La alternativa consiste en eliminar un error utilizando la instrucción ZOOM y corrigiendo el error pixel por pixel, lo que en el caso de una caja o un disco mal colocados lleva cierto tiempo. Una corrección que sería muy bien recibida sería la de restringir el alcance de la instrucción OOPS a la última pulsación del botón Select en vez de a la última salida del menú principal.

Como uno esperaría de un dispositivo que se conecta en la puerta para palanca de mando, se puede utilizar la Koala-pad como una palanca de estas características, permitiendo a los usuarios acceder a la Koala-pad desde sus propios programas. La posición del cursor se puede obtener desde BASIC leyendo (PEEK) las posiciones 54297 y 54298 para las coordenadas X e Y respectivamente.

Las pantallas se pueden guardar en disco y trans-

ferir fácilmente a los propios programas del usuario, permitiendo desarrollar aventuras estilo *Hobbit* con texto en la parte inferior de la pantalla y una imagen arriba. Mediante la utilización de la Koala-pad es posible guardar y recuperar hasta 16 pantallas distintas de ocho Kbytes en un disco. Si bien no es posible cargar una pantalla desde disco directamente en la memoria de pantalla, en la guía para el usuario se ha incluido un programa que permite la transferencia de pantallas desde la zona en la cual están cargadas a la memoria de pantalla.

La resolución máxima de una pantalla Koala Painter guardada es de 255×255 pixels si está incluida en un programa en BASIC, si bien utilizando código máquina se pueden conseguir resoluciones mayores. Esta limitación podría plantear problemas, dado que la pantalla en alta resolución del Commodore 64 en realidad tiene unas dimensiones de 320×200 pixels. La Koala-pad está limitada a un máximo de 255 pixels porque éste es el número mayor que puede direccionar un byte; para direccionar una pantalla completa se requerirían direcciones de 16 bits. Lamentablemente, esto puede llevar a que el usuario pierda una porción de la visualización. Esto es así porque cuando la pantalla no está siendo visualizada, estará almacenada en algún lugar dentro de la RAM del usuario. Por lo tanto, la pantalla de ocho Kbytes, junto con la información de color, se debe transferir desde su posición en la memoria para el usuario hasta la memoria de pantalla en alta resolución, que empieza en la posición 55296.

KOALA-PAD

DIMENSIONES
210×165 mm

PANTALLA
Se utiliza la visualización en pantalla completa del Commodore 64, de 320×200, si bien ésta se puede ver restringida a 255×255 al llamar una pantalla desde un programa en BASIC

INTERFACE
Se conecta a través de la puerta para palanca de mando del Commodore 64

DOCUMENTACION
Si bien la mayor parte de la información requerida está en los manuales del usuario y del programador, hay muy poco texto de explicación

VENTAJAS
Le permite al usuario del Commodore 64 producir visualizaciones de gráficos en alta resolución directamente, sin programar

DESVENTAJAS
La resolución de la tablilla es deficiente y difusa en algunas modalidades



Creación de imágenes

Esta serie de imágenes muestra muchas de las características de la Koala-pad. En primer lugar, utilizando la instrucción ZOOM diseñamos cada una de las letras de la palabra *pleasure* (placer). Éstas se pueden situar en cualquier lugar de la pantalla; luego, empleando la instrucción COPY, las podemos colocar en la parte inferior de la pantalla en la secuencia correcta. A continuación, utilizando la instrucción LINE trazamos una línea a través de la pantalla en la parte superior de las letras y dibujamos la silueta de la estrella para producir un efecto tridimensional. Se agregan bloques de color haciendo uso de la instrucción FILL. Por último, se pueden incluir características adicionales empleando las instrucciones DISC y CIRCLE

Ian McKinnell

En dos ruedas

El juego que presentamos es muy simple y consta de sólo 35 líneas de BASIC, pero aun así resulta muy atractivo

El juego se llama *En su moto* y está basado en una escena de la película *Tron*, producida por la firma Walt Disney. Es una contienda entre dos contrincantes que exige destreza y reacciones rápidas y tiene lugar en un circuito cerrado. Cada jugador tiene una motocicleta que viaja a una velocidad desbocada y no se la puede detener. Su único control permite girar 90º a la velocidad máxima. Estas motocicletas dejan tras de sí paredes sólidas de luz, y el objetivo del juego consiste en forzar al contrincante a estrellarse con el laberinto que uno va creando mientras se desplaza a toda velocidad.

El juego se ha implementado en el ZX Spectrum, que no se destaca por la velocidad de su BASIC. Dado que se trata de un juego de acción, el programa se ha diseñado atendiendo más a la velocidad que a la elegancia, por lo que gran parte del listado puede parecer algo desestructurado.

Como el rayo

El atractivo de *En su moto* es que a pesar de que la idea es simple, el juego requiere destreza y concentración. Crear trampas y barreras con la huella de la moto, evitando al mismo tiempo chocar y estrellarse con la del rival es extraordinariamente difícil, incluso a la poca velocidad que permite el BASIC del Spectrum

Ian McKinnell

Se han evitado llamadas a subrutinas y otros dispositivos estructurados, porque habrían supuesto sacrificar velocidad de ejecución.

La primera etapa es el diseño del circuito y de la visualización del marcador. Como puede verse, éste es bastante sencillo, lo que contribuye a la brevedad del programa final. El único punto que hay que destacar es que el borde del circuito corresponde a un carácter dentro de la zona de pantalla utilizable. Esto es para asegurar que los gráficos resultantes de una colisión con la pared del circuito no se salgan de pantalla:

```
10 LET p=0:LET q=0
100 BORDER 0:PAPER 0:CLS
110 PRINT AT 0,1;INK 6;"Moto uno=";q
120 PRINT AT 0,19;INK 5;"Moto dos=";p
130 INK 2
140 PLOT 8,8:DRAW 239,0:DRAW 0,159
150 DRAW −239,0:DRAW 0,−159
```

El circuito se ha dibujado en rojo, y hemos elegido el amarillo para representar a la motocicleta uno y el cyan (azul claro) para la motocicleta dos. Las variables *p* y *q* retienen el marcador en curso de los dos rivales.

La siguiente etapa es la inicialización de las variables, y aquí debemos empezar a pensar acerca de cómo implementaremos la acción principal del juego. La acción para una única motocicleta es bastante directa y está reflejada en el diagrama de flujo (p. 1113). Utilizando POINT verificamos si la posición en curso de la moto está ocupada y, de ser así, pasamos a la rutina de colisión. Si no lo está, pasamos a esa posición mediante PLOT y después leemos el teclado para comprobar cualquier cambio de dirección. Nuestra posición se incrementa, entonces, en uno en la dirección actual, y el ciclo vuelve a comenzar. Por tanto, necesitamos dos variables: dos para las coordenadas *x* e *y* en curso, y dos para la dirección en curso a lo largo de los ejes *x* e *y*.

No obstante, se trata de dos motos que se mueven al mismo tiempo. Una solución elegante sería utilizar cuatro matrices de dos elementos *x(2)* e *y(2)*, para las posiciones, por ejemplo, pero esto retardaría el juego, de modo que tenemos que emplear ocho variables separadas:

```
200 LET x=40:LET y=88
210 LET m=215:LET n=88
220 LET a=1:LET b=0
230 LET i=−1:LET j=0
```

Esto establece las posiciones iniciales de las motos y hace que se muevan a un pixel por vez. La acción básica es, por tanto, bastante fácil de implementar:

```
400 IF PONT(x,y)=1 THEN LET col=6:GOTO 700
410 IF POINT (m,n)=1 THEN LET col=5:LET x=m:
    LET y=n:GOTO 700
420 PLOT INK 6;x,y:PLOT INK 5;m,n
```

(entre las líneas 500 y 570 está la rutina de teclado que establece nuevos valores para *a, b, i* y *j*):

```
600 LET x=x+a:LET y=y+b
610 LET m=m+i:LET n=n+j
620 GOTO 400
```

Quizá el único punto que se preste a confusión sea la línea 410, donde se establecen para la moto número dos las variables de la moto número uno, y se introduce una nueva variable, col. Ello es así para que se pueda utilizar una única rutina para la acción de la colisión, donde *x* e *y* se emplean simplemente para indicar el punto en el cual tiene lugar la colisión, y col establece el color.

La rutina para verificar el teclado ha de ser rápida, pero lamentablemente no hemos tenido más remedio que utilizar sentencias IF...THEN que son más bien lentas. No obstante, podemos utilizar la veloz instrucción IN para leer las teclas. Las teclas de control elegido son Q y A, que controlan el movimiento hacia arriba y hacia abajo de la moto uno, y P y ENTER para la moto dos. Izquierda y derecha son X y C para la moto uno, y N y M para la moto dos. (Véase p. 846 para una explicación completa acerca de cómo se relacionan los bloques de teclas con los bytes de la memoria.)

```
500 IF IN 64510=190 THEN LET a=0:LET b=1
510 IF IN 65022=190 THEN LET a=0:LET b=-1
520 IF IN 65278=187 THEN LET a=-1:LET b=0
530 IF IN 65278=183 THEN LET a=1:LET b=0
540 IF IN 57342=190 THEN LET i=0:LET j=1
550 IF IN 49150=190 THEN LET i=0:LET j=-1
560 IF IN 32766=187 THEN LET i=1:LET j=0
570 IF IN 32766=183 THEN LET i=-1:LET j=0
```

Todo lo que resta es la rutina de la colisión y la actualización de los marcadores. Se eligió una serie de círculos concéntricos que se van ampliando, centrados en el punto del impacto, con radios de cuatro, seis y ocho pixels:

```
700 FOR d=1 TO 3
710 CIRCLE BRIGHT 1;INK col;x,y,2+d*2
720 NEXT d
730 IF col=6 THEN LET p=p+1:GOTO 750
740 LET q=q+1
750 GOTO 100
```

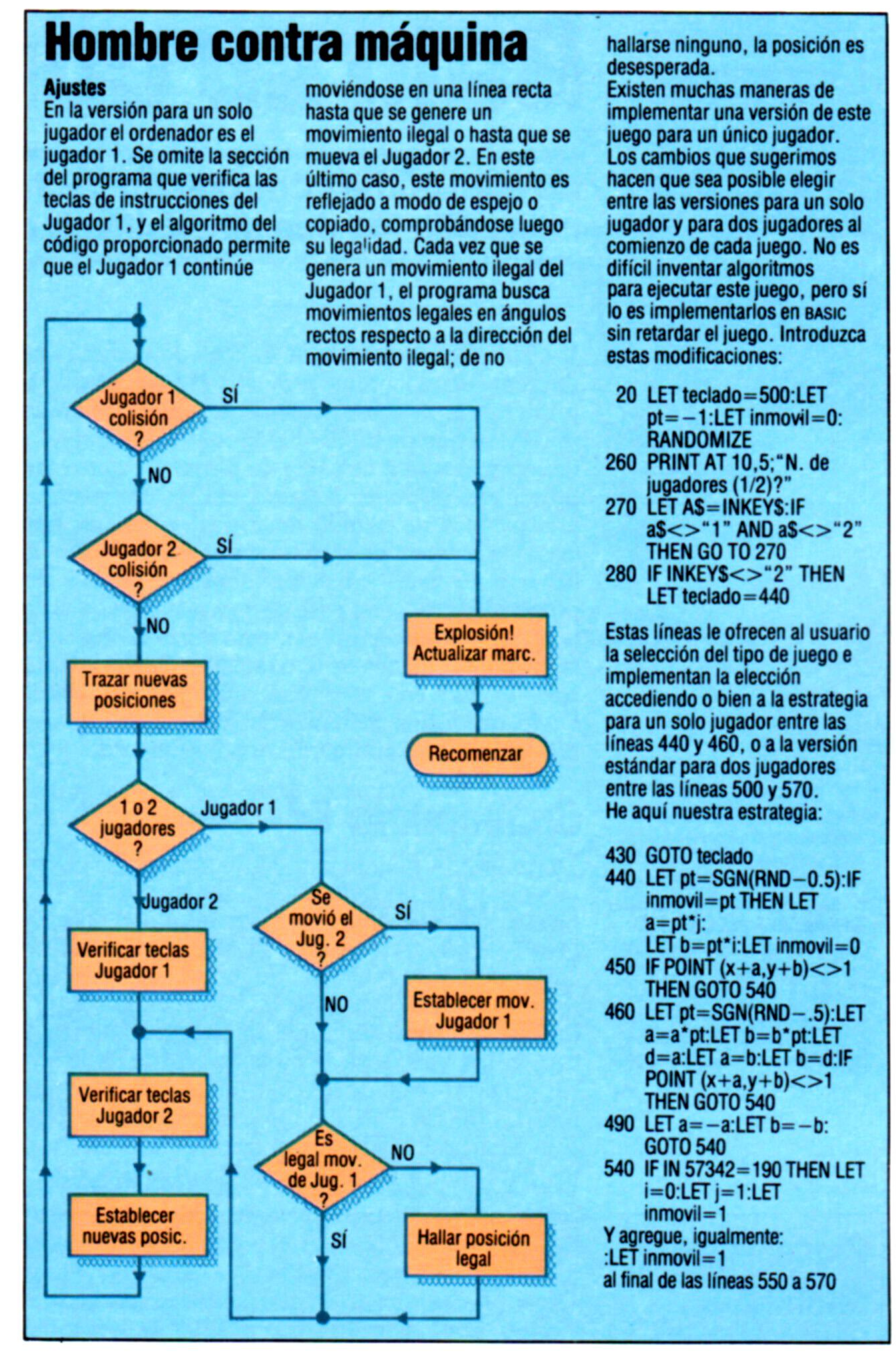
Con esto se acaba el juego, con la última sentencia saltando hacia atrás hasta los procedimientos de inicialización del principio. Al juego, sin embargo, le iría bien un procedimiento de inicio para facilitar más su utilización:

```
300 PRINT AT 10,5;INK 7;"PULSE CUALQUIER
    TECLA PARA EMPEZAR"
310 IF INKEY$=""THEN GOTO 310
320 PRINT AT 10,5;"                    "
```

Esto proporciona una pausa entre rondas consecutivas. Sólo resta guardar el juego en cassette, utilizando de preferencia SAVE"En su moto"LINE10, de modo que el juego se ejecute automáticamente apenas se cargue.

Evidentemente se podría hacer que el juego fuera más emocionante, con pantallas de instrucciones, pantallas de carga, una opción para un solo jugador con una rutina de estrategia que controle la otra motocicleta del modo que hemos sugerido, sonido y mejores gráficos. Pero estas últimas opciones harán que el juego se desarrolle de manera excesivamente lenta.

En un próximo capítulo volveremos a escribir *En su moto* en lenguaje máquina con el fin de demostrar su auténtico potencial.



Hombre contra máquina

Ajustes

En la versión para un solo jugador el ordenador es el jugador 1. Se omite la sección del programa que verifica las teclas de instrucciones del Jugador 1, y el algoritmo del código proporcionado permite que el Jugador 1 continúe moviéndose en una línea recta hasta que se genere un movimiento ilegal o hasta que se mueva el Jugador 2. En este último caso, este movimiento es reflejado a modo de espejo o copiado, comprobándose luego su legalidad. Cada vez que se genera un movimiento ilegal del Jugador 1, el programa busca movimientos legales en ángulos rectos respecto a la dirección del movimiento ilegal; de no hallarse ninguno, la posición es desesperada.

Existen muchas maneras de implementar una versión de este juego para un único jugador. Los cambios que sugerimos hacen que sea posible elegir entre las versiones para un solo jugador y para dos jugadores al comienzo de cada juego. No es difícil inventar algoritmos para ejecutar este juego, pero sí lo es implementarlos en BASIC sin retardar el juego. Introduzca estas modificaciones:

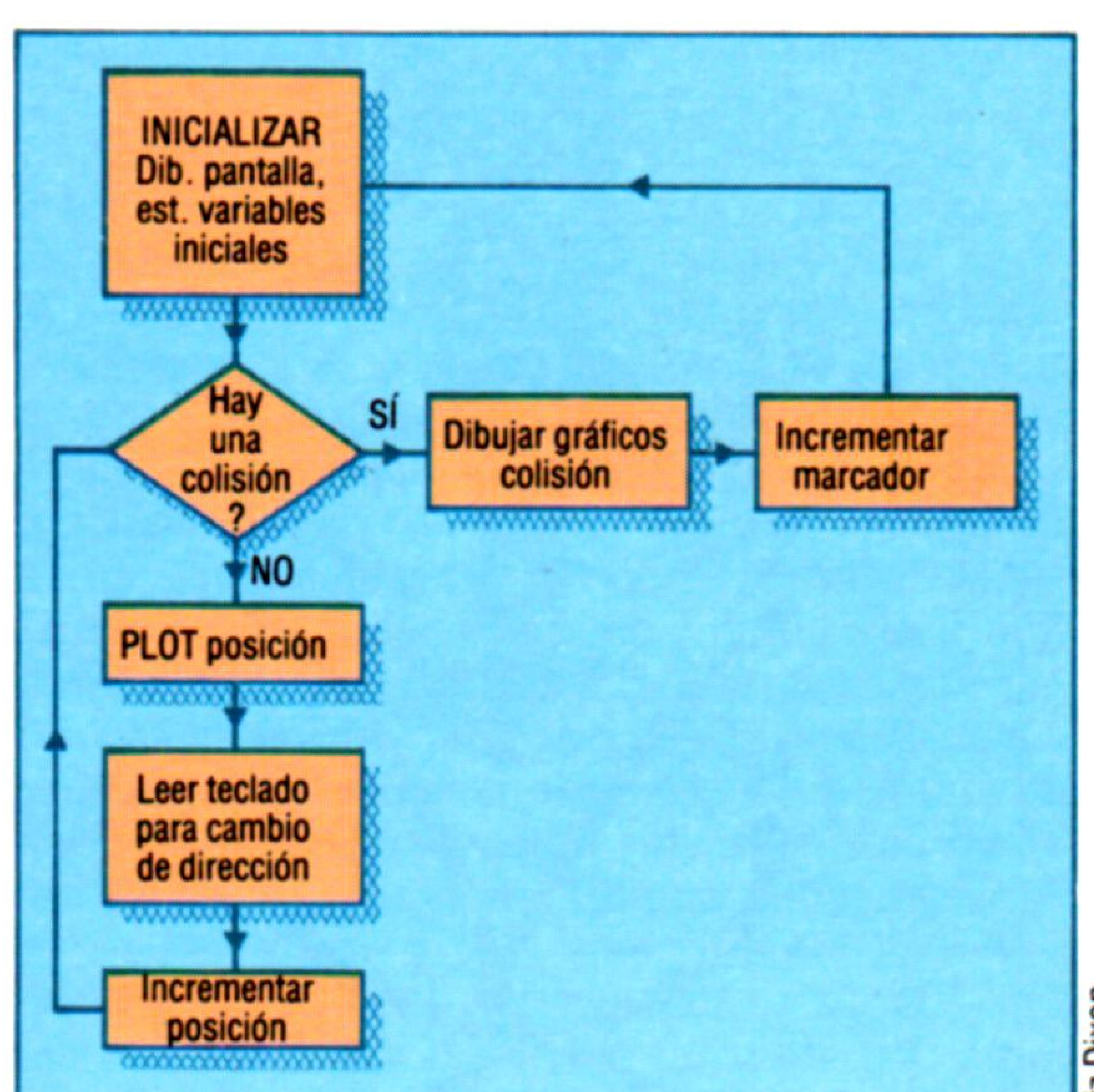
```
 20 LET teclado=500:LET
    pt=-1:LET inmovil=0:
    RANDOMIZE
260 PRINT AT 10,5;"N. de
    jugadores (1/2)?"
270 LET A$=INKEY$:IF
    a$<>"1" AND a$<>"2"
    THEN GO TO 270
280 IF INKEY$<>"2" THEN
    LET teclado=440
```

Estas líneas le ofrecen al usuario la selección del tipo de juego e implementan la elección accediendo o bien a la estrategia para un solo jugador entre las líneas 440 y 460, o a la versión estándar para dos jugadores entre las líneas 500 y 570.

He aquí nuestra estrategia:

```
430 GOTO teclado
440 LET pt=SGN(RND-0.5):IF
    inmovil=pt THEN LET
    a=pt*j:
    LET b=pt*i:LET inmovil=0
450 IF POINT (x+a,y+b)<>1
    THEN GOTO 540
460 LET pt=SGN(RND-.5):LET
    a=a*pt:LET b=b*pt:LET
    d=a:LET a=b:LET b=d:IF
    POINT (x+a,y+b)<>1
    THEN GOTO 540
490 LET a=-a:LET b=-b:
    GOTO 540
540 IF IN 57342=190 THEN LET
    i=0:LET j=1:LET
    inmovil=1
```

Y agregue, igualmente:
:LET inmovil=1
al final de las líneas 550 a 570



Para un solo jugador
Este diagrama de flujo simplificado ilustra la estructura del programa con un solo jugador. Cada proceso se repite en el juego en versión para dos jugadores

Bajo su control

He aquí cómo controlar un coche Lego con una palanca de mando, utilizando los dos dispositivos creados en capítulos anteriores

Las palancas de mando se utilizan de manera muy diferente en el Commodore 64 y el BBC Micro. El primero emplea una palanca de mando estándar tipo Atari que opera mediante cuatro interruptores direccionales y un pulsador de disparo. Contrastando con esta disposición digital, el BBC Micro utiliza una palanca de mando analógica, o paleta. Esta clase de palanca no se basa en el simple hecho de hacer o no hacer contactos, sino que emplea dos potenciómetros, uno para el movimiento izquierda-derecha y el otro para el movimiento arriba-abajo (si bien una palanca de tipo digital se puede adaptar para su uso con la puerta analógica del BBC). Dado que estos métodos de funcionamiento son tan distintos, los abordaremos por separado.

Commodore 64

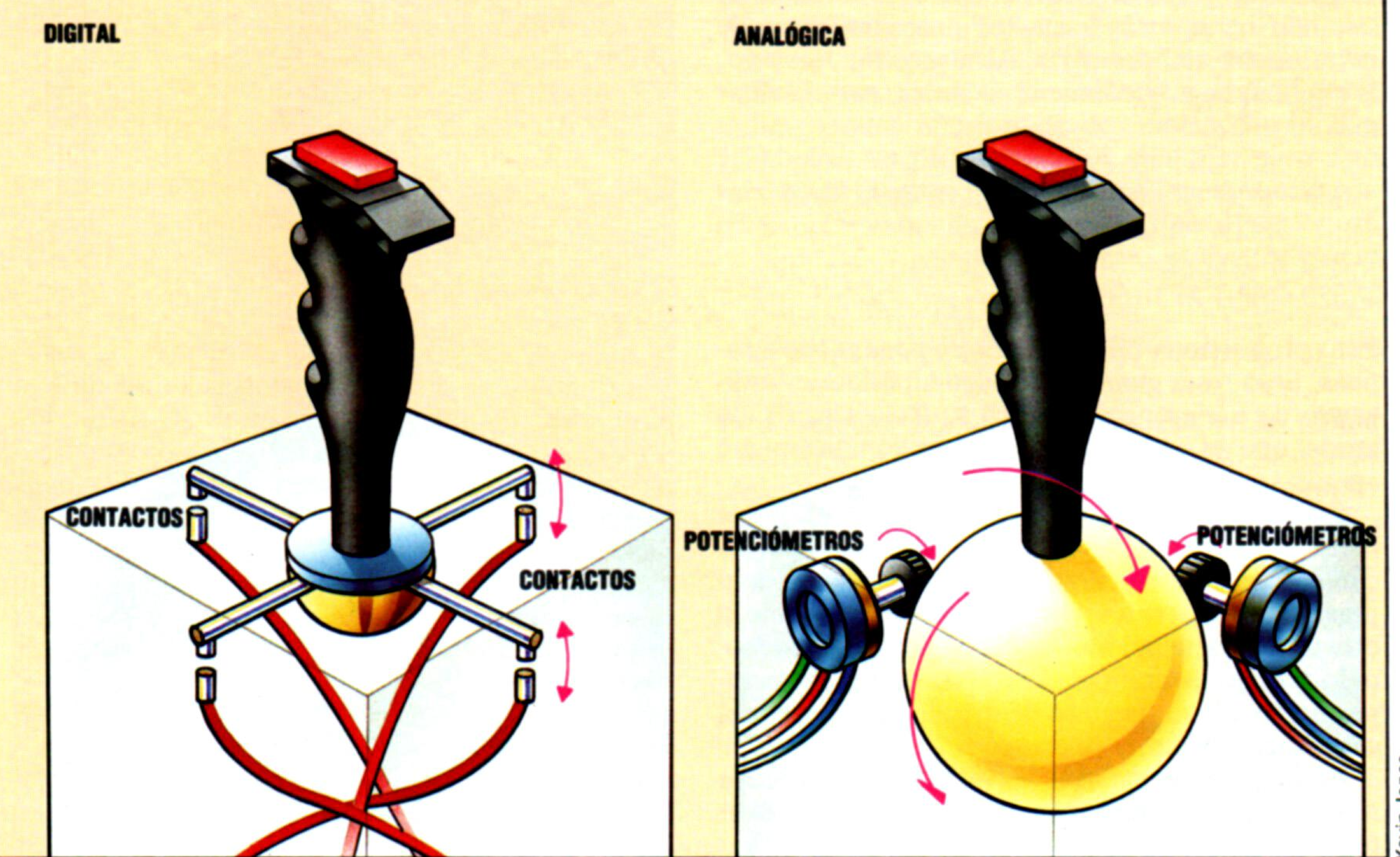
La palanca de mando tipo Atari que utiliza el Commodore 64 se enchufa en una de las puertas para juegos, situada junto al interruptor de potencia on/off. En la explicación y el programa siguientes emplearemos la puerta 2, de modo que si dispone de una palanca de mando enchúfela en la puerta 2 (la que está más cerca del interruptor on/off). Si ésta se desplaza desde la posición central hacia arriba, se cerrará uno de los cuatro interruptores internos. La puerta 2 se une directamente con la posición de memoria 56320, y el cierre de este interruptor hará que uno de los bits de esta posición se ponga a 0, más o menos de la misma forma en que el cerrar un interruptor externo conectado a la caja buffer haría que un bit del registro de datos de la puerta para el usuario se pusiera a 0. Entre el breve programa que reseñamos a continuación, que visualiza repetidamente el valor del registro de datos. Con el programa en marcha desplace la palanca de mando hacia uno y otro lado y accione el pulsador de disparo, observando las diversas modificaciones que se producen en el valor visualizado en la pantalla.

```
10 REM **** LEER PALANCA MANDO CBM 64 ****
20 PUERTA2=56320
30 PALANCA=PEEK(PUERTA2):GOSUB500
40 PRINT CHR$(145);PALANCA,B$
50 GOTO30
60 :
500 REM S/R CONVERSION A BINARIO
510 B$="":N=PALANCA
520 FOR D=1 TO 8
530 N1=INT(N/2):R=N-2*N1
540 B$=B$+STR$(R):N=N1
550 NEXT D
560 RETURN
```

Al cabo de algunos minutos de experimentación ya debería resultar claro qué bits de la posición de la palanca de mando corresponden a los cuatro interruptores de dirección y al pulsador de disparo. Normalmente, con la palanca en la posición central, el contenido de la posición de ésta es 127, es decir, 01111111. Al empujar la palanca, el valor pasa a 126 (01111110). El bit 0 está relacionado con el interruptor de dirección hacia arriba. Los efectos

Procedimientos diferentes

La palanca de mando digital, o "de contacto", simplemente registra el movimiento del bastón en dos direcciones cualesquiera de las cuatro principales; la cantidad de movimiento no se mide. La palanca de mandos analógica emplea potenciómetros perpendicularmente opuestos para medir el movimiento del bastón, y comunica la información como dos voltajes variables. Éstos se convierten luego en un número mediante el convertidor de analógico a digital





de los interruptores de la palanca en la posición de memoria de ésta son los siguientes:

Palanca de mando	Decimal	Binario
Central	127	01111111
Arriba	126	01111110
Abajo	125	01111101
Izquierda	123	01111011
Derecha	119	01110111
Disparo	111	01101111

Quizá también haya observado que el desplazamiento de la palanca en diagonal hace que dos interruptores se cierren simultáneamente. A pesar de que para controlar nuestro vehículo no necesitamos la detección del movimiento en diagonal, los resultados de tales desplazamientos son éstos:

Palanca de mando	Decimal	Binario
Arriba e izquierda	122	01111010
Arriba y derecha	118	01110110
Abajo e izquierda	121	01111001
Abajo y derecha	117	01110101

El siguiente programa utiliza una palanca de mando para controlar los movimientos del vehículo de motores gemelos. El vehículo se deberá conectar a la caja de salida de la misma forma que en la página 1066, y la palanca se deberá enchufar en la puerta para juegos 2, situada en el lado derecho del ordenador.

```
10 REM PALANCA DE MANDO CBM 64
20 RDD=56579:REGDAT=56577
30 POKERDD,255:REM TODAS SALIDA
40 PALANCA=PEEK(56320):REM PALANCA PUERTA 2
50 GOSUB1000:REM COMPROBAR PALANCA DE MANDO
60 POKEREGDAT,0:GOTO40
90 :
1000 REM S/R COMPROBACION PALANCA MANDO
1005 IF PALANCA=127 THEN POKEREGDAT,0
1010 IF PALANCA=126 THEN POKEREGDAT,5
1020 IF PALANCA=125 THEN POKEREGDAT,10
1030 IF PALANCA=123 THEN POKEREGDAT,6
1040 IF PALANCA=119 THEN POKEREGDAT,9
1050 IF PALANCA=111 THEN POKEREGDAT,0:END
1060 RETURN
```

BBC Micro

La palanca de mando del BBC es un dispositivo analógico que se basa en dos potenciómetros para proporcionar información sobre los movimientos efectuados hacia arriba-abajo e izquierda-derecha. La diferencia fundamental entre una palanca de mando digital y una de tipo analógico como la que emplea el BBC Micro reside en que la última proporciona información acerca de la "posición" entre límites dados, mientras que la primera sólo proporciona información acerca de la dirección del recorrido. Los potenciómetros trabajan así:

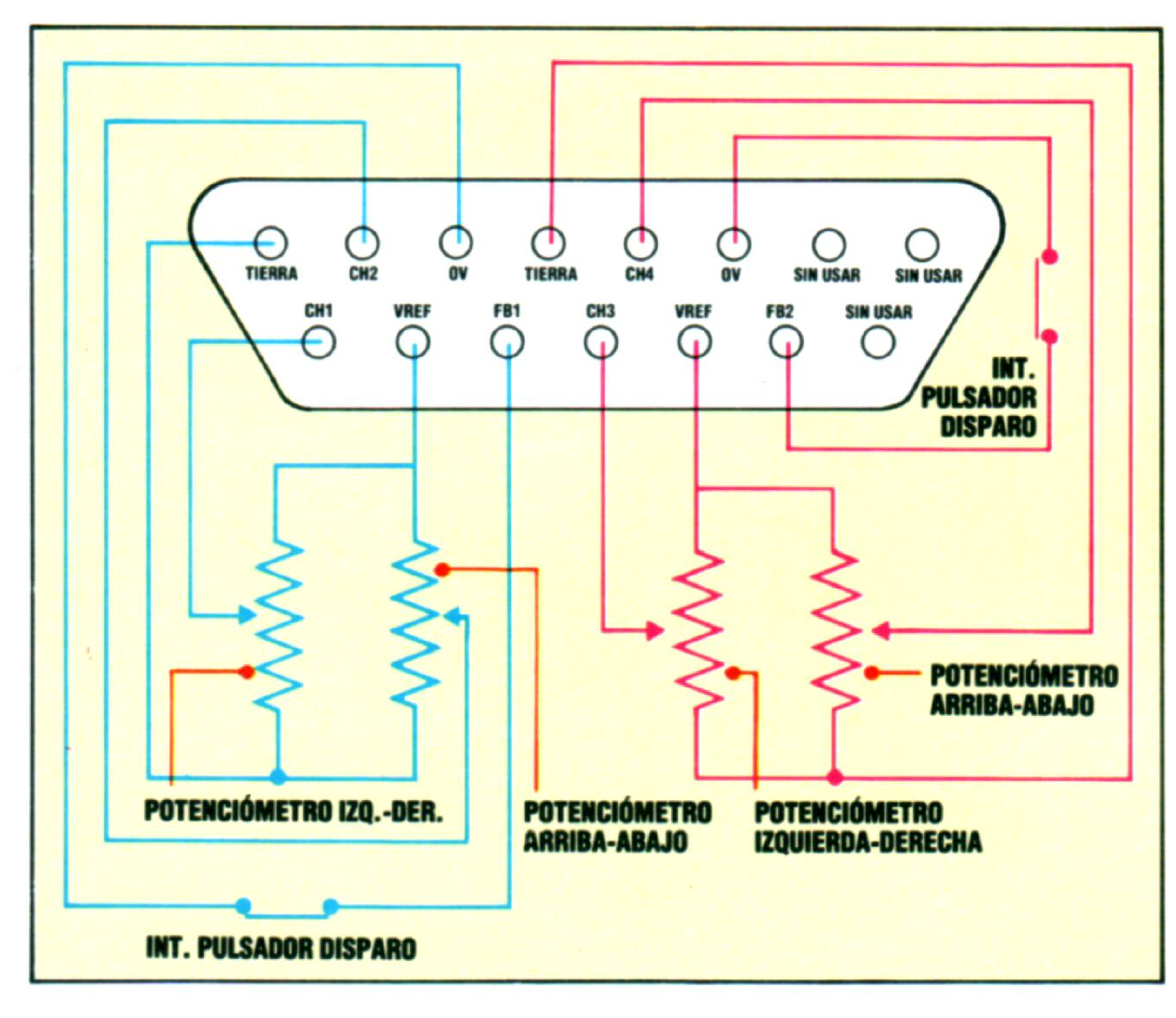
Un potenciómetro es básicamente una resistencia a través de la cual se aplica un voltaje. Un tercer conector del potenciómetro se puede desplazar a lo largo de la resistencia tomando una fracción del voltaje suministrado. Esta fracción depende de la posición de la conexión móvil. En un tipo lineal de potenciómetro, si la conexión móvil se colocara a medio camino a través de la resistencia, los voltajes derivados serían la mitad del voltaje suministrado. Por consiguiente, mediante el desplazamiento de la conexión central se puede obtener cualquier voltaje entre cero y el voltaje suministrado. Girar el mando del volumen de una radio es, esencialmente, mover la conexión del medio a través de la resistencia del potenciómetro de control de volumen. En una palanca analógica el movimiento de la conexión del medio se realiza moviendo la paleta.

Las palancas de mando para el BBC por lo general se venden a pares. En el diagrama superior vemos las conexiones de la puerta analógica del BBC Micro para las palancas 1 y 2.

El micro proporciona un voltaje de referencia a través de cada potenciómetro y el voltaje derivado de la conexión del medio se alimenta a través de las entradas de dos canales. El canal 1 se utiliza para la entrada desde el potenciómetro izquierda-derecha, y el canal 2 para aceptar la entrada desde el potenciómetro arriba-abajo. El pulsador de disparo es un simple interruptor para abrir o cerrar contacto.

Una vez se han aceptado las entradas del potenciómetro, se las debe convertir de la forma analógica a la digital mediante un convertidor interno. Esta conversión se realiza comparando el voltaje de entrada con el voltaje de referencia, y el tiempo de conversión es de alrededor de 10 milisegundos para la lectura de cada canal. Después de que la señal de entrada de la palanca está en forma digital, podemos utilizar los valores para controlar el vehículo.

La entrada a la puerta analógica se puede leer desde BASIC mediante el empleo de la instrucción ADVAL. El valor que devuelve ADVAL está comprendido entre 0 y 65520, correspondiendo el límite superior a una entrada igual al voltaje de referencia. Las reducciones del voltaje de entrada irán produciendo números correspondientemente más pequeños hasta que un voltaje de entrada de cero voltios produzca la devolución por parte de ADVAL de un valor de cero. Para nuestra aplicación sólo interesan los dos valores límite. El canal leído por ADVAL está determinado por el número entre paréntesis que sigue a la palabra clave. Por tanto, ADVAL(1) leerá el canal 1 y devolverá un valor que estará comprendido entre 0 y 65520.



Reflexiones analógicas
Las palancas de mando para el BBC Micro por lo general se suministran por parejas, que van a un único conector. Las salidas de las patillas de la puerta analógica, tal como se ven desde fuera de la máquina, muestran que la palanca 2 está conectada de modo similar a la palanca 1. En el lugar de cualquiera de las dos se pueden conectar palancas individuales

ADVAL(0) cumple dos funciones diferentes. Los dos bits menos significativos corresponden a los pulsadores de disparo de la palanca de mando 1 y la palanca de mando 2. X=ADVAL(0) AND 3 devolverá un valor de uno si se acciona el pulsador de disparo de la palanca 1. X=ADVAL(0) DIV 256 dará el número del canal que ha completado en último lugar una conversión de analógico a digital (A a D).

Dado que la conversión de cada canal de entrada analógica lleva alrededor de 10 milisegundos, el proceso de cada uno de los cuatro canales llevará 40 milisegundos. En nuestra aplicación sólo empleamos los canales 1 y 2. Podemos reducir el tiempo de conversión utilizado especificando qué canales requieren conversión. Esto se puede hacer por medio de *FX16,2, que habilita los canales 1 y 2 pero inhabilita los canales 3 y 4. El siguiente programa combina toda esta información.

```
10 REM CONTROL DE PALANCA BBC
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=255:REM TODAS SALIDA
40 REM HABILITAR CANALES A-D 1 y 2
50 *FX16,2
60 REPEAT
70 PROCleer—palanca
80 UNTIL disparo=1
90 END
100 :
110 DEF PROCleer—palanca
120 REPEAT
130 canal=ADVAL(0) DIV 256
140 UNTIL canal<>0:REM ESPERAR CONVERSION
150 IF canal=1 THEN PROCizquierda—derecha
160 IF canal=2 THEN PROCarriba—abajo
170 ENDPROC
180 :
190 DEF PROCizquierda—derecha
200 REPEAT
210 valpal=ADVAL(1)
220 IF valpal<100 THEN ?REGDAT=9
230 IF valpal>64000 THEN ?REGDAT=6
240 disparo=ADVAL(0) AND 3
250 PRINT?REGDAT,canal,valpal
260 UNTIL(valpal>100 AND valpal<64000) OR disparo=1
270 ?REGDAT=0
280 ENDPROC
290 :
300 DEF PROCarriba—abajo
310 REPEAT
320 valpal=ADVAL(2)
330 IF valpal<100 THEN ?REGDAT=10
340 IF valpal>64000 THEN ?REGDAT=5
350 disparo=ADVAL(0) AND 3
360 PRINT?REGDAT,canal,valpal
370 UNTIL(valpal>100 AND valpal<64000) OR disparo=1
380 ?REGDAT=0
390 ENDPROC
```

Respuestas a los ejercicios

1) La calibración de su vehículo se puede efectuar cronometrando el tiempo invertido para recorrer varias distancias, comúnmente 10 cm, 20 cm, 50 cm, 100 cm y 150 cm. Calculando la velocidad sobre cada distancia y haciendo un promedio, se puede obtener una buena estimación para la distancia recorrida en un segundo. Este valor se puede utilizar para controlar el vehículo sobre distancias medidas. Se podría adoptar un enfoque similar para girar, seleccionando una cantidad de ángulos y efectuando cronometrajes. Controlar motores encendiéndolos y apagándolos durante períodos de tiempo medidos puede plantear muchas dificultades, sin contar con que la estructura del programa controlador es tal que los intervalos se deben medir con la mayor exactitud. Diferencias de centésimas de segundo pueden producir enormes divergencias en las distancias o los ángulos de giro.

2) El listado del programa ofrecido en la página 1093 permitirá conducir el vehículo a través del recorrido con obstáculos. Seguir el patrón en sentido contrario es un poco más complicado. Primero debemos asignar un par de variables para cada dirección junto con su inverso. De modo que, por ejemplo, adelante y marcha atrás se agrupan en el mismo par.

Commodore 64

```
17 A(1)=5:B(1)=10:A(2)=10:B(2)=5
18 a(3)=6:B(3)=9:A(4)=9:B(4)=6
```

Entonces se pueden agregar las siguientes rutinas para reproducir la secuencia grabada al revés:

```
92 GOSUB 2000: REM REPRODUCCION INVERTIDA
2000 S/R REPRODUCCION INVERTIDA
2010 FOR I=C TO 1 STEP -1
2020 FOR J=1 TO 4
2030 IF DR(I,1)=A(J) THEN POKE REGDAT,B(J):J=4
2040 NEXT J
2050 T=TI
2060 IF (TI-T)<DR(I,2) THEN 2060
2070 NEXT I
2080 STOP
2090 RETURN
```

BBC Micro

```
1020 DIM DR(100,2),A(10),B(10)
1025 A(1)=5:B(1)=10:A(2)=10:B(2)=5
1026 A(3)=6:B(3)=9:A(4)=9:B(4)=6

1115 PROCreproduccion—invertida

2000 DEF PROCreproduccion—invertida
2010 FOR I=C TO 1 STEP -1
2020 FOR J=1 TO 4
2030    IF DR(I,1)=A(J) THEN?REGDAT=B(J):J=4
2040 NEXT J
2042 TIME=0
2045 REPEAT UNTIL TIME>=DR(I,2)
2047 ?REGDAT=0
2050 NEXT I
2055 PRINT"PULSE C PARA CONTINUAR"
2060 REPEAT A$=GET$
2070 UNTIL A$="C"
2080 ENDPROC
3000 FORI=1TOC:PRINTDR(I,1),DR(I,2)
3010 NEXT
```

3) Suponiendo que la línea 7 es adelante, la línea 6 es atrás, la 5, izquierda y la 4, derecha:

```
10 REM INTERRUPTORES EXTERNOS BBC
20 RDD=&FE62:REGDAT=&FE60
30 ?RDD=15:REM LINEAS 4-7 ENTRADA
40 ?REGDAT=0
50 PROCleer—entrada
60 GOTO 50
70 :
80 DEF PROCleer—entrada
90 IF(?REGDAT AND 240)=240 THEN ?REGDAT=0
100 IF(?REGDAT AND 128)=0THEN ?REGDAT=5
110 IF(?REGDAT AND 64)=0THEN ?REGDAT=10
120 IF(?REGDAT AND 32)=0THEN ?REGDAT=6
130 IF(?REGDAT AND 16)=0THEN ?REGDAT=9
140 ENDPROC
```

```
10 REM INTERRUPTORES EXTERNOS CMB 64
20 RDD=56579:REGDAT=56577
30 POKERDD,15:REM LINEAS 4-7 ENTRADA
40 POKEREGDAT,0:REM MOTORES APAGADOS
50 REM L7 ADEL,L6 ATR,L5 IZQ,L4 DER
55 REM ESTA BAJA UNA DE LAS LINEAS DE ENTRADA?
60 IF(PEEK(REGDAT)AND240)<>240 THENGOSUB1000:GOTO60
70 POKEREGDAT,0:GOTO60
80:
1000 REM S/R EXPLORACION LINEAS ENTRADA
1005 IF(PEEK(REGDAT)AND128)=0THENPOKEREGDAT,5
1010 IF(PEEK(REGDAT)AND64)=0THENPOKEREGDAT,10
1020 IF(PEEK(REGDAT)AND32)=0THENPOKEREGDAT,6
1030 IF(PEEK(REGDAT)AND16)=0THENPOKEREGDAT,9
1040 RETURN
```



Saltos y tablas

Presentamos un análisis del direccionamiento indexado

Ante todo hay que dejar bien sentado que el direccionamiento indirecto no es un modo más de direccionar sino un recurso adicional que se emplea juntamente con casi todos los demás modos; en realidad se trata de un paso ulterior en el cálculo de la dirección efectiva (la dirección de donde efectivamente se toman los datos). La dirección efectiva o real se calcula de alguna de las maneras que hemos descrito, pero si se escoge el direccionamiento indirecto implica que el contenido de la dirección así calculada es considerado a su vez como una dirección, junto con la siguiente posición consecutiva de la memoria. Esta dirección se convierte en dirección efectiva final, y de ella se extraen los datos.

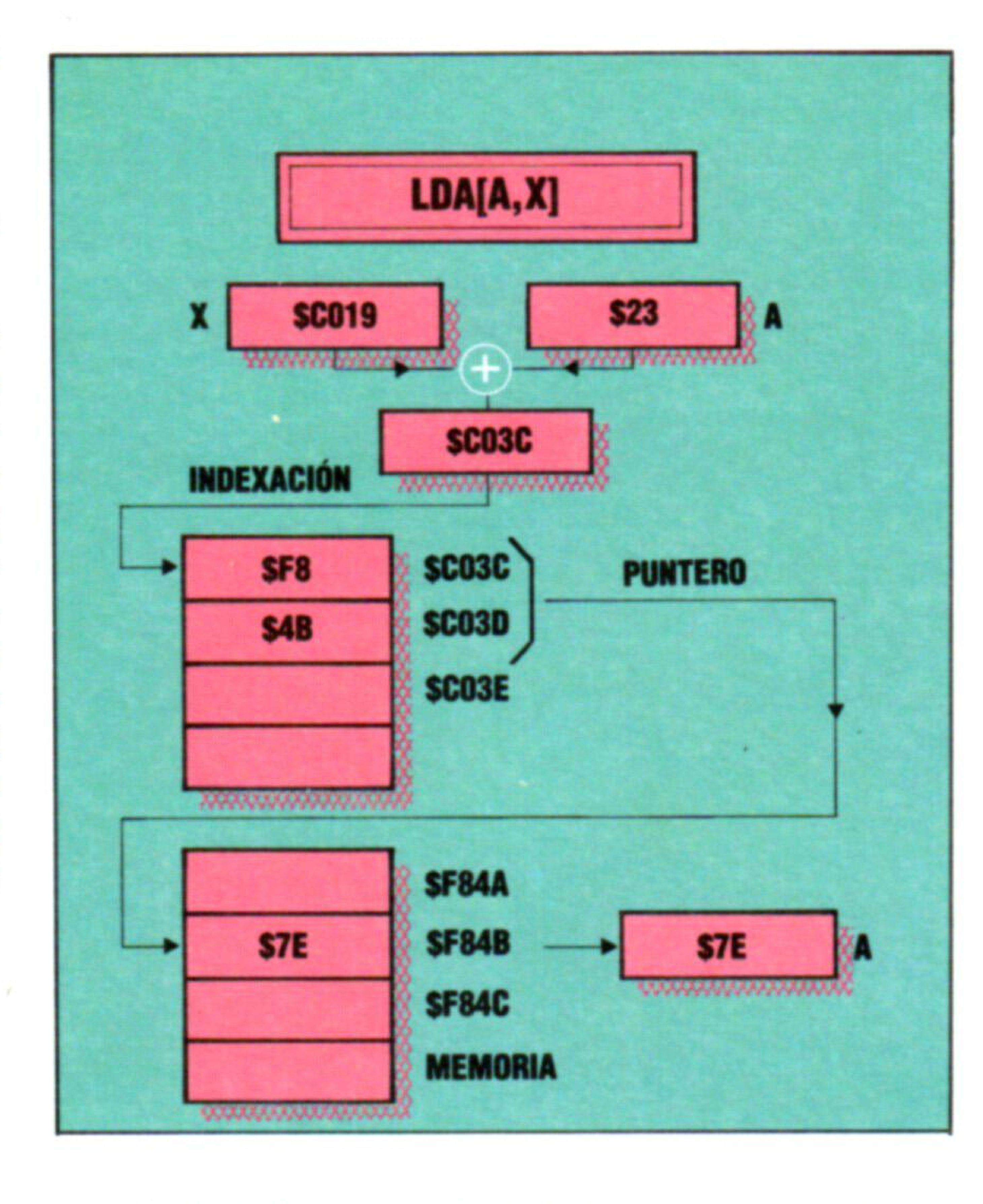
Si los siguientes valores son almacenados:

Dirección	Contenido
3000	40
3001	0A
400A	F2

entonces la instrucción LDA $3000 cargará el valor $40 en el acumulador A, siendo la dirección efectiva $3000. El modo indirecto siempre queda especificado por medio de corchetes que encierran el operando, así LDA [$3000] cargará el valor $F2 en A, y la dirección efectiva es el valor almacenado en la dirección que a su vez quedó almacenado en $3000 y $3001, en nuestro caso, $400A. Es de notar que para el 6809 se conviene en que las direcciones sean almacenadas con el byte *hi* precediendo al byte *lo*: así $40 se almacena en $3000 y $0A en $3001. Es lo que se conoce por el convenio *hi-lo*. Justo el convenio opuesto al empleado por el Zilog Z80 y el MOS Tech 6502: en éstos el byte *lo* de la dirección, $0A, se almacenaría en $3000 y el byte *hi*, $40, en $3001.

El direccionamiento indirecto a menudo se potencia si se usa con el indexado. La instrucción LDA [A,X], que está en el modo indexado indirecto, calculará primero una dirección sumando los contenidos de A y de X y después utilizará el valor de 16 bits almacenado en esta posición y la siguiente como una dirección cuyo contenido se cargará en A.

De hecho, el 6809 usa menos el direccionamiento indirecto que muchos procesadores (los programas tanto para el 6502 como para el Z80 lo utilizan profusamente) dada la potencia de sus modos de direccionamiento indexados. Pero hay ocasiones en que el modo indirecto resulta útil. Una de ellas, que nos disponemos a emplear largamente en un próximo capítulo, es el uso de dispositivos de interface de periféricos. Los procesadores Motorola, al contrario que las familias 8080 y 8086 del Intel, tienen una E/S expresada como un mapa en la memoria. Los registros de comunicaciones introducidos en los dispositivos de interface aparecen dentro del mapa principal de la memoria del sistema, y los valores pueden almacenarse o sacarse de ellos como si fueran unas posiciones de memoria cualesquiera en vez de ser, efectivamente, un conducto hacia el dispositivo de la interface. Una rutina para controlar uno de estos dispositivos (p. ej., una rutina de impresión) necesita la dirección del registro de la interface. Si el dispositivo se reubica en el mapa de memoria, o si no hay más que un dispositivo de ese tipo, entonces es mucho más sencillo tratarlo por medio del cambio de una posición de memoria, la que contiene la dirección del registro de comunicación del dispositivo (un puntero al dispositivo), mejor que recurrir a encontrar y cambiar cada ocurrencia de su dirección. La rutina referencia el dispositivo indirectamente, por medio del puntero.



Direccionamiento indexado indirecto
El argumento [A,X] de la instrucción LDA está entre corchetes, lo que equivale a decir que el contenido de X (aquí,$C019) será añadido al contenido de A ($23), obteniéndose así una dirección de 16 bits ($C03C). Este byte y el siguiente ($C03D) serán tratados como un puntero a la dirección efectiva ($F84B) cuyo contenido se carga finalmente en A. Dado que X se suma a A antes del acceso indirecto, se habla de un direccionamiento indirecto preindexado. La otra alternativa, el postindexado, exige el cálculo de la dirección indirecta previo a la indexación

Este ejemplo muestra el uso general del direccionamiento indexado. Cuando las direcciones a las que se refiere un programa pueden ser cambiadas, es preferible emplear punteros de dirección fija para señalar estas posiciones. De esta manera, los cambios en las posiciones actuales sólo exigen cambios en el contenido de los punteros.

El uso más común de esta técnica consiste en la estructura conocida como *tabla de salto (jump table)*, que no es más que una tabla de punteros. Todo sistema operativo contiene un gran número de valiosas rutinas que realizan las funciones elementales de la máquina. Por ejemplo, la lectura de un carácter del teclado o la visualización de un carácter en la pantalla. Muchos programas en código máquina tendrán que valerse de estas rutinas en uno u otro momento. En muchos casos, se podrá acceder a ellas gracias a una tabla de salto, lo que quiere decir que estas rutinas apuntadas por las ta-

Ian McKinnell

Cortesía de Air Call

Acceso indirecto

A pesar de que el direccionamiento indirecto es fundamental en las operaciones de un ordenador, resulta difícil encontrar ejemplos análogos en la vida real. Algo se parece, sin embargo, al servicio que prestan los listines de teléfono. Si alguien desea llamar a un abonado, no lo llama directamente pues puede hallarse en cualquier lugar. En su lugar, pide a información que le indique la dirección e incluso que le ponga en contacto. En este ejemplo, el servicio de la centralita proporciona un acceso indirecto (o direccionamiento) a sus abonados

blas de salto pueden ser cambiadas o reubicadas en memoria sin que ello implique la modificación de los programas que las utilizan. Cuando se diseña una nueva versión del sistema operativo o se actualiza la ROM, no suelen respetarse las primitivas posiciones de estas rutinas; pero si la tabla de salto permanece en su posición, con los punteros modificados de modo que reflejen las nuevas direcciones, cualquier software escrito para el antiguo sistema operativo puede ejecutarse en el nuevo.

Una técnica muy común que se emplea en muchos sistemas operativos consiste en retener un punto de entrada y realizar todas las llamadas de subrutina a esta única dirección. Uno de los registros de la CPU se suele añadir para proporcionar un código de función, que sirve para determinar qué subrutina es la llamada. Este código es empleado como un índice o desplazamiento dentro del vector correspondiente de la tabla de salto, pasando el control a través de este puntero a la rutina deseada.

Como ejemplo, supongamos que tenemos cuatro Kbytes de ROM, colocados en $F000, de los cuales los primeros 256 bytes (desde el $F000 hasta el $F0FF) contienen una tabla de hasta 128 direcciones de subrutinas almacenadas en la ROM. La rutina de entrada (la dirección a través de la cual se direccionan todas las demás rutinas del SO) se encuentra en $F100 y espera recibir en el acumulador B un código de función con un valor entre 0 y 127. Este código se emplea por la rutina de entrada para pasar el control a las subrutinas apropiadas y devolverlo después al programa que las ha llamado una vez finalizada su ejecución. La rutina de llamada para la función número 1 es:

LDB #1	coloca el código de la función en B
JSR $F100	llama a la subrutina de entrada

En sí misma, la subrutina de entrada sería:

LDX $F000	dirección del inicio de la tabla de salto
LSLB	desplaza B un lugar a la izquierda (*left shift*: como si multiplicara el contenido de B por dos), dado que toda entrada en la tabla tiene dos bytes de longitud. De esta manera, el puntero apropiado para el código 1 de función se almacena en $F002 y $F003, mientras que el indicador del código 2 está en $F004 y $F005, etc.
BRA [B,X]	pasa el control a la dirección encontrada en la posición número B de la tabla

Obsérvese cómo el paso a la rutina se realiza con BRA (o JMP) y no con BSR (o JSR); esto es así para que la RTS, al final de la rutina de SO, devuelva el control directamente al programa desde donde se realizó la llamada en vez de retornar a la rutina de entrada.

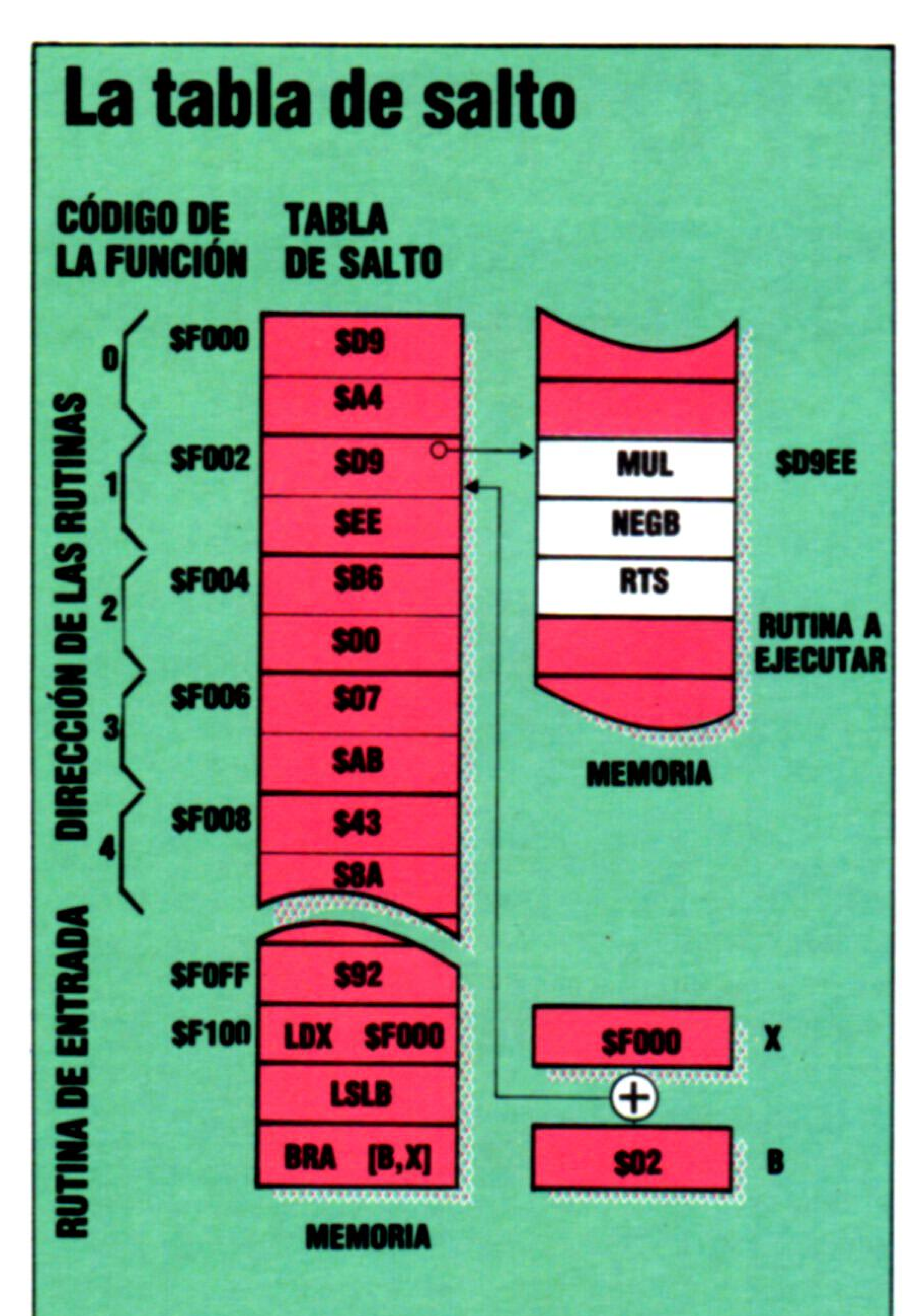
Nuestro ejemplo siguiente muestra otro uso del direccionamiento indirecto cuando se trata con una pantalla cuyo mapa de visualización se tiene en memoria; en muchos micros la memoria de pantalla ocupa un sector de la memoria principal y puede accederse a ella directamente en caso de que se requiera mayor velocidad. Supongamos, simplificando, que la pantalla ocupa un bloque de memoria que va desde el $E000 hasta el $E3FF, representando 16 líneas de 64 caracteres. La posición del cursor es un valor de 16 bits dentro de esos límites y se encuentra en $E400. La primera subrutina borra la pantalla escribiendo el carácter de espacio (código 32 en ASCII) sobre cada posición de los 16×64 caracteres. La segunda subrutina va a imprimir el carácter contenido en A sobre la pantalla y en el lugar que ocupa en este momento el cursor, a menos que el carácter sea precisamente el de "re-



torno de carro" (13 en ASCII), en cuyo caso borrará lo que queda de línea y posicionará el cursor al comienzo de la línea siguiente. Éste se representa por el carácter subrayado ("—").

```
SPACE   EQU 32          Código ASCII de espacio
CR      EQU 13          Código ASCII de retorno de carro
HOME    EQU $E000       Comienzo de la memoria de pantalla
LENGTH  EQU 1024        Tamaño de la memoria de pantalla
                        (16 líneas×64 caracteres=1024)
CURSOR  EQU $E400       $E400 y $E401 juntos apuntan a la
                        posición actual del cursor en el área
                        de memoria de pantalla
        ORG $1000
CURCHR  FCB 95          Carácter subrayado (ASCII 95)
***Subrutina para borrar la pantalla***
        LDA #SPACE      Carácter espacio en A
        LDX #HOME       Sitúa el cursor al inicio de la pantalla
                        Almacena en CURSOR la posición
        STX CURSOR      actual de éste (o sea, $E400,$E401)
        LDB #LENGTH     Introduce en B el tamaño de la
                        pantalla
LOOP1   STA [CURSOR]    Deposita un espacio en la posición
                        actual del cursor
        INC CURSOR      Incrementa la posición del cursor
        DECB            Decremento de la cantidad de
                        memoria que queda entre la posición
                        del cursor y el final de la memoria
        BGT LOOP1       Espacio siguiente hasta agotar toda
                        la memoria de pantalla
        STX CURSOR      El cursor vuelve a su posición inicial
        LDA CURSOR      Introduce el carácter del cursor en A
                        (código ASCII)
        STA [CURSOR]    Almacena el carácter del cursor en la
                        posición actual de éste
        RTS
***Subrutina para visualizar el carácter guardado en A, si es
visible***
        CMPA SPACE      El espacio en blanco es el primer
                        carácter imprimible en ASCII
        BLT NOTP        Si el acumulador contiene un valor
                        ASCII menor que 32, no se puede
                        visualizar: GOTO NOTP
        STA [CURSOR]    Almacena la posición actual del
                        cursor
        INC CURSOR      Incrementa la posición del cursor
CHKEOS  LDX #HOME       Comprueba el final de la pantalla
        LEAY #LENGTH,X  Fin de pantalla introducido en Y
        CMPY CURSOR     Si la posición del cursor excede el
                        final de la pantalla entonces…
        BGT FINISH      se alcanzó el final de la pantalla, por
                        lo tanto GOTO FINISH
***Subrutina para desplazar la pantalla***
SCROLL  LEAY 64,X       Y es la longitud de una línea tomada
                        de X (final de la memoria de pantalla)
        LDB #LENGTH     Calcula la cantidad a desplazar
        SUBB #64        Resta 64 a la longitud
LOOP 2  LDA,Y+          Mueve los caracteres una línea hacia
                        atrás (nótese el autoincremento,
                        véase p. 1098)
        STA,X+
        DECB
        BGT LOOP2       Bucle hasta acabar el desplazamiento
        LDD CURSOR      Cursor en el comienzo de la última
                        línea
        SUBD #64
        STD CURSOR
        BRA FINISH
***Subrutina para comprobar si hay "retorno de carro"***
NOTP    CMPA #CR        Este carácter no visualizable es el de
                        "retorno de carro"?
        BNE FINISH      Si no es así, ignorarlo
        LDD CURSOR      Se puede investigar cómo esto
        ANDB            obtiene el inicio de la línea siguiente
        #%11100000      (basta analizar la máscara AND)
        ADDD #64
        STD CURSOR
        BRA CHKEOS      Comprueba el final de la pantalla
FINISH  LDA CURSOR      Se introduce en A el carácter del
                        cursor
        STA [CURSOR]    Lo almacena en la posición actual del
                        cursor
        RTS
```

La tabla de salto



La tabla de salto de este ejemplo consiste en una lista de 128 punteros de dirección de dos bytes, situados entre $F000 y $F0FF. Cada uno de estos punteros contiene la dirección de inicio de una rutina situada, a su vez, en cualquier sitio de la memoria. Para ejecutar cualquiera de estas rutinas necesitamos tan sólo cargar el acumulador B con el código de la función ($01, p. ej.) que sirve para identificar la rutina (en este ejemplo, situada en $D9EE) y después se realiza un salto JSR hacia la denominada "rutina de entrada", empezando aquí en la dirección $F0FF. Supondremos que estas rutinas están en ROM (dado que forman parte del software situado en ROM, tal como el sistema operativo), de manera que podemos consultar el código de la función y la dirección de inicio de la rutina de entrada en el manual del programador.
La rutina de entrada multiplica el código de la función por dos y lo emplea como desplazamiento respecto a la dirección de inicio de la tabla para encontrar el puntero deseado de la dirección de la rutina: el indicador de la rutina $01 está colocado, por ejemplo, en $F002 (=$F000+2×$01), el puntero de la rutina $02 está en $F004 (=$F000+2×$02), y así sucesivamente. El puntero es entonces empleado por la rutina de entrada en una instrucción de bifurcación indirecta para pasar el control a la rutina de turno colocada en $D9EE. Obsérvese que la rutina de entrada bifurca (más que llama) a la rutina de ejecución, de modo que cuando se encuentra RTS, el control se devuelve al punto del programa en que la rutina de entrada fue llamada por primera vez.
La ventaja de una tabla de salto está en que permite a los programadores volver a diseñar y a colocar las rutinas direccionadas por la tabla, y que los programas escritos antes de estas revisiones pueden ser ejecutados en el nuevo sistema.

El origen de las especies

Atari ha producido versiones del conocido juego "Pacman" (Comecocos) para numerosos micros. Veamos cómo funciona en el Spectrum y el Vic-20

Pacman (Comecocos) fue el juego recreativo de laberinto original, y estableció la fórmula que desde entonces ha seguido el software para ordenadores personales como *Atic Atac* y *Jet Set Willy*. En tales juegos el personaje central debe viajar a través de una red de corredores o habitaciones, recogiendo, mientras tanto, tesoros y sorteando los diversos peligros que lo acechan durante su camino. Bajo ciertas condiciones, es posible que el jugador invierta los papeles de los monstruos atacantes; en *Sabre Wulf*, por ejemplo, tropezar con ciertas orquídeas hace que el jugador sea invisible para las criaturas de la selva. Esta idea se tomó prestada de los juegos de aventuras, en los cuales la posesión de una espada mágica o un objeto similar coloca al jugador en una condición ventajosa.

En *Pacman*, el personaje que da nombre al juego se coloca en el centro de un laberinto, y el jugador debe intentar guiar esta figura alrededor de la pantalla mientras va tragándose los puntos que iluminan el camino. Se debe mantener el ojo atento a los fantasmas perseguidores, quienes exhiben una tozuda determinación a atrapar a Pacman en las diversas calles sin salida que abundan en el laberinto. La ingestión de una "píldora de energía" permite al jugador perseguir y comerse a los fantasmas, consiguiendo de este modo puntos extras, y unos bocados de frutas que van apareciendo al azar contribuyen, si se los come, a aumentar el marcador.

Establecer comparaciones entre distintas versiones del mismo juego podría parecer injusto; al fin y al cabo, un juego diseñado para una determinada máquina obviamente aprovechará mejor las capacidades de ese micro. Esto es así en especial para los juegos diseñados específicamente para las máquinas Atari, que disfrutan de merecida fama, aunque son caros. No obstante, cabe decir que las nuevas versiones de *Pacman* no están a la altura de la versión original.

En los ordenadores personales Atari, *Pacman* configura un enorme laberinto con excelentes gráficos y sonido, movimiento uniforme de los sprites y varios niveles de destreza, y salvando el hecho de que el juego no se ejecute en una máquina exclusiva, esta versión es una réplica fidedigna del juego recreativo. Las versiones para el Spectrum y el Commodore no están en el mismo nivel. En el caso del Vic-20, los programadores se enfrentaron al problema habitual de tener que hacer caber un cuarto de litro en una jarra de un octavo de litro: el laberinto es de sólo una cuarta parte del tamaño de su equivalente Atari, mientras que los sprites son el doble de grandes de lo que eran originalmente, y en este espacio tan restringido se hace difícil esquivar a los fantasmas. No obstante, los gráficos están bien definidos, el movimiento es uniforme y el sonido es tan bueno como el de la versión Atari. En la versión para el Spectrum se proporcionan instrucciones más completas y se le ofrece al jugador la opción de control por teclado en vez de utilizar las palancas de mando necesarias para las versiones Atari y Commodore. Es muy poco lo que los programadores de Atarisoft podrían haber hecho con las limitadas facilidades de sonido del Spectrum, pero los gráficos son sumamente decepcionantes; a pesar de que el laberinto es bastante parecido al del original, el inestable movimiento hace que jugar al *Pacman* sea algo parecido a mirar una película de cine mudo. Las limitaciones de la versión para el Vic se explican en virtud de las restricciones propias del hardware, pero resulta difícil entender por qué la versión para el Spectrum es tan pobre.

Cuando el *Pacman* hizo su primera aparición, en 1980, se convirtió muy rápidamente en una sensación, pero en la actualidad parece un poco anticuado. Hace apenas unos años, Atarisoft podría haber vendido miles de ejemplares de *Pacman*, pero hoy en día existen en el mercado muchos juegos recreativos mejores.

Pacman: Para todos los ordenadores Atari, el Commodore Vic-20 y el Spectrum
Editado y distribuido por: Atari, AUDELEC, Compás de la Victoria, 3, 29012 Málaga, España
Autores: Atari
Palanca de mando: Necesaria
Formato: Atari y Vic-20, cartucho; Spectrum, cassette



Editorial Delta, S.A.



10056

9 788485 822836